Optimisation:

De l'estimation paramétrique à l'apprentissage, une ballade entre théorie et pratique

S. Delprat

Chapitre 2 – Optimisation sans contrainte

Chapitre 2 – Optimisation sans contrainte

1) Contexte
2) Algorithme à base de Gradient
3) Méthode de Newton & quasi-Newton
4) Méthode « Régions de confiance » (Trust region)

Contexte

Soit f une fonction convexe de classe $\mathcal{C}^2$.

On cherche le minimum (global) de la fonction f: $\min_{x \in \mathbb{R}^n} f(x)$

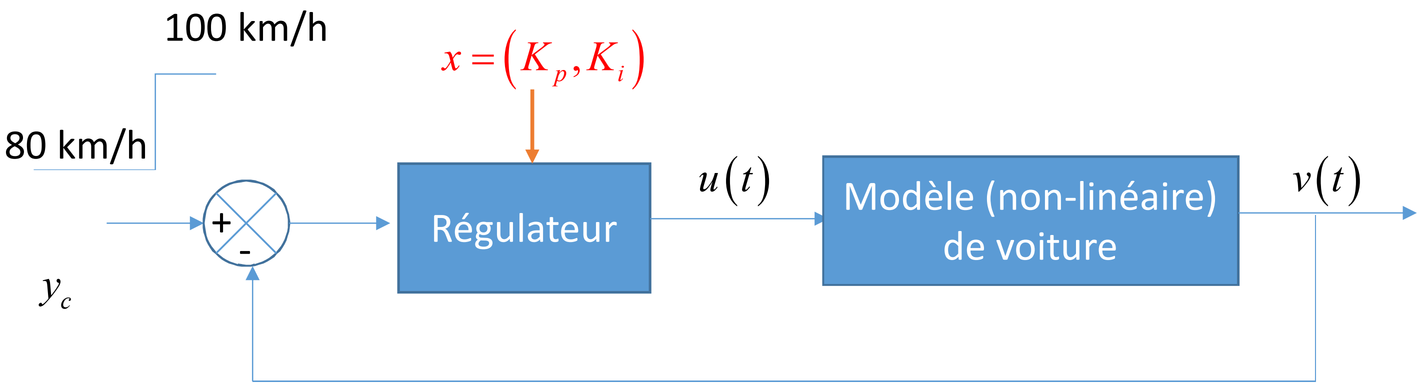
Nb: en pratique, si f n'est pas convexe, on trouvera un des minimum locaux

Problème:

On peut évaluer la fonction f et son gradient mais on ne connait pas la fonction f.

Contexte

<u>Problème:</u> Trouver les paramètres d'un régulateur de vitesse.



$u(t)$: couple délivré par le moteur thermique

$v(t)$: vitesse du véhicule

<u>Objectif:</u> minimiser :

1. L'erreur de vitesse $\left| y_c(t) - v(t) \right|$
2. La consommation d'énergie $u(t)^2$

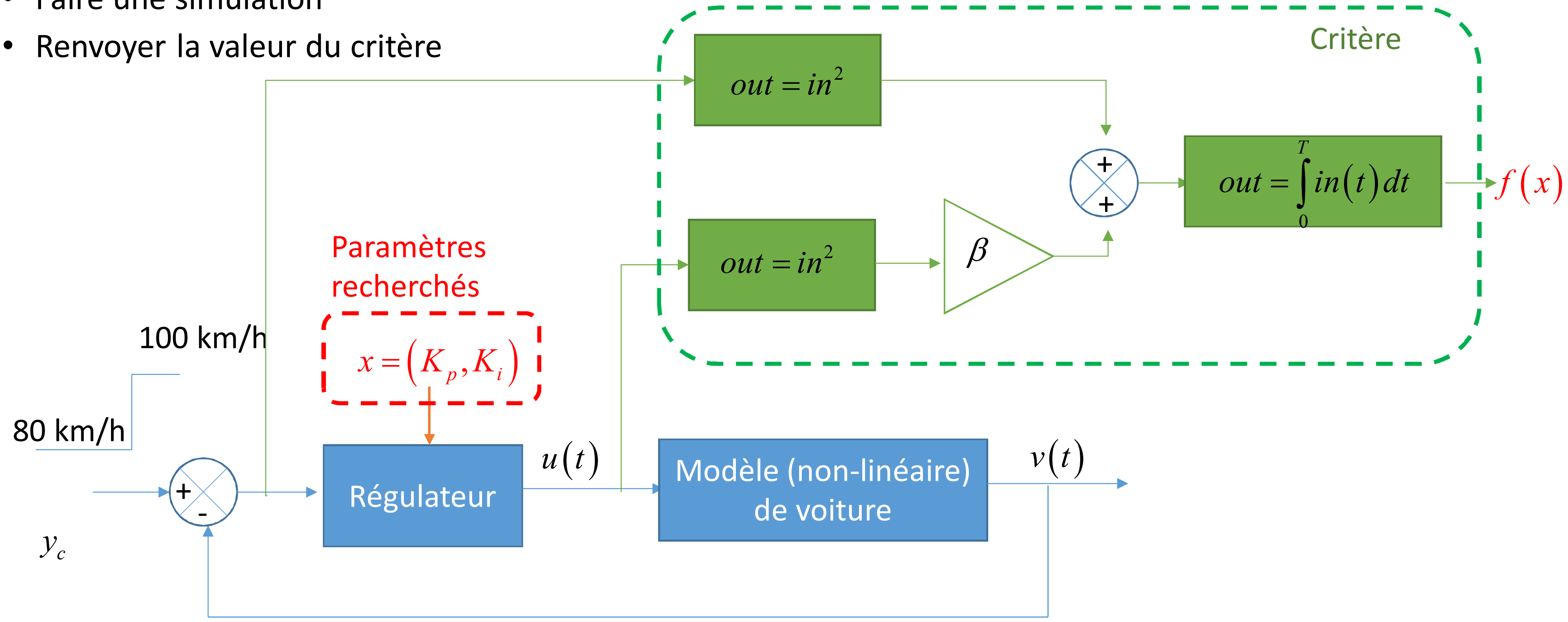
Compromis conso-perfo

Attention: on veut obtenir le minimum sur une durée donnée [0,T]: $f(x) = \int_0^T \left[\left(y_c(t) - v(t) \right)^2 + \beta u(t)^2 \right] dt$

Contexte

Evaluer la fonction $f(x)$, c'est:

- Fixer les valeurs de x
- Modifier les paramètres K_p,K_i du modèle en conséquence
- Faire une simulation
- Renvoyer la valeur du critère



Contexte

Soit f une fonction convexe de classe $\mathcal{C}^2$.

On cherche le minimum (global) de la fonction f: $\min_{x \in \mathbb{R}^n} f(x)$

Nb: en pratique, si f n'est pas convexe, on trouvera un des minimum locaux

Problème:

On peut évaluer la fonction f et son gradient mais on ne connait pas la fonction f.

Solution:

On essaye de construire une suite de valeur qui converge vers le minimum global (ou local si f n'est pas convexe).

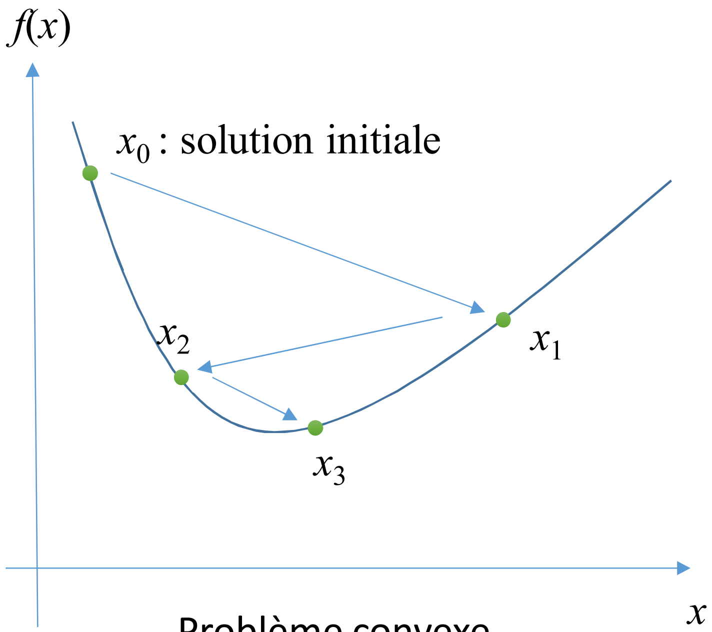
Problème : Il faut un point de départ.

Contexte

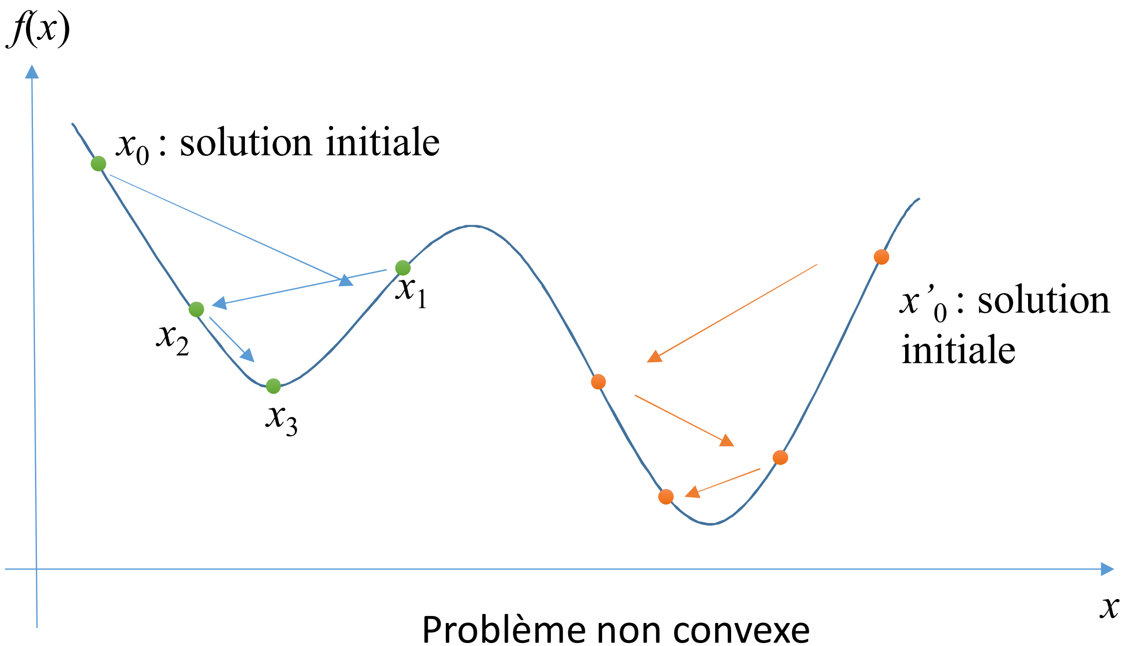
Soit f une fonction convexe de classe $\mathcal{C}^2$.

On cherche le minimum (global) de la fonction f: $\min_{x \in \mathbb{R}^n} f(x)$

Nb: en pratique, si f n'est pas convexe, on trouvera un des minimum locaux



Problème convexe



Problème non convexe

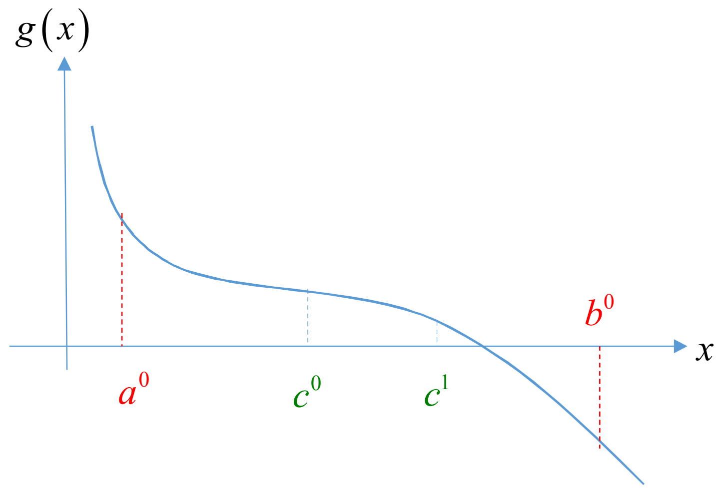
Préambule: Recherche d'un zéro d'une fonction

- **Méthode de bi-section**

Soit une fonction g monotone (au moins localement) $g : \mathbb{R} \rightarrow \mathbb{R}$

On cherche une racine $g\left(x^*\right) = 0$

On connait 2 bornes a^0 et b^0 encadrant la solution : $g\left(a^0\right) < 0 < g\left(b^0\right)$



Algorithme

k=0

Répéter:

- On coupe l'intervalle en 2: $c^k = \dfrac{a^k + b^k}{2}$

- Si $g(c^k)$>0 ALORS

$$a^{k+1} = c^k$$
$$b^{k+1} = b^k$$

SINON

$$a^{k+1} = a^k$$
$$b^{k+1} = c^k$$

Fin SI

On garde l'intervalle qui contient la racine

Jusqu'à $b^k - a^k < \varepsilon$

Préambule: Recherche d’un zéro d’une fonction

- Méthode de bi-section

Savoir-faire n°1 : Calcul Matlab

Savoir trouver avec une précision donnée le zéro d’une fonction monovariable sur un intervalle donné

```
clear all;
close all;
clc;

% Algorithme de bisection
f=@(x) (x-3).^3;

x=linspace(0,5,200);
figure;plot(x,f(x),'r')
grid on
hold on;
racine=bisection(f,-10,10,1e-5);
plot(racine,f(racine),'b*');

function c=bisection(f,a,b,eps)
ended=0;
while ended==0
    c=(a+b)/2;
    fc=f(c);
     if fc>0
        b=c;
    else
        a=c;
    end
    ended=(b-a)<eps;
end
end
```

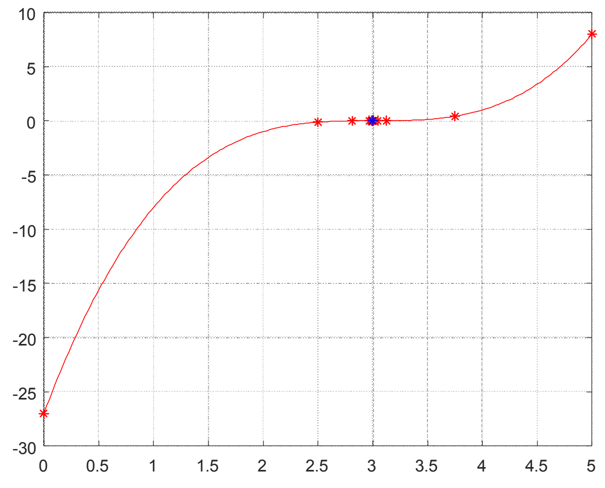
Intérêt :

- Simple à mettre en œuvre
- Pas besoin de connaitre la dérivée de f

Inconvénient:

- Convergence lente

$$f(x)=(x-3)^3$$



Direction de recherche

Construction d'une série qui converge vers un minimum local de f :

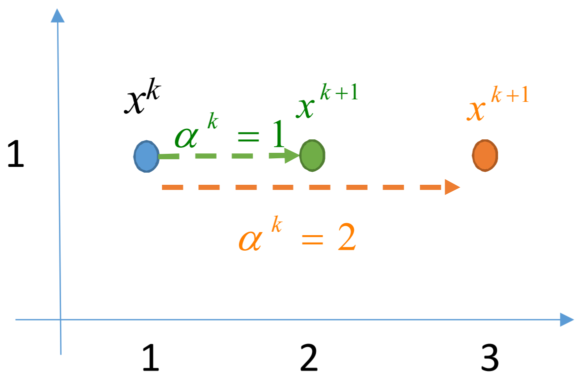
x_0 : point de départ, fournit par l'utilisateur ou aléatoire

$$x^{k+1} = x^k + \alpha^k d^k$$

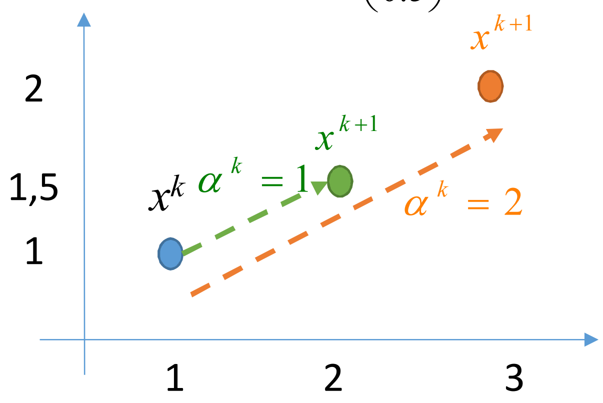
$d^k \in \mathbb{R}^{n,1}$: Direction de recherche

$\alpha^k \in \mathbb{R}$: Taille du pas de recherche

Direction: $d^k = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$



Direction: $d^k = \begin{pmatrix} 1 \\ 0.5 \end{pmatrix}$



Vocabulaire: direction de descente

Soit $x \in \mathbb{R}^n$, $w \in \mathbb{R}^n \setminus \{0\}$ est une direction de descente en x s'il existe α_0>0 tel que

$$f(x+\alpha d) \leq f(x),\ \forall \alpha [0, \alpha_0]$$

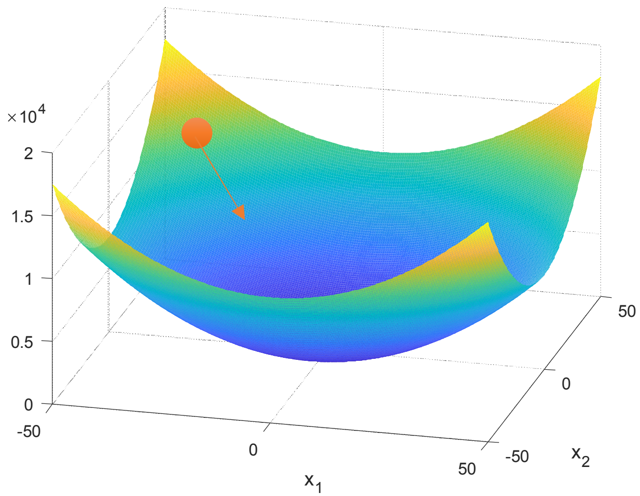
Soit $x \in \mathbb{R}^n$, $w \in \mathbb{R}^n \setminus \{0\}$ est une direction de descente *stricte* en x s'il existe α_0>0 tel que

$$f(x+\alpha d) < f(x),\ \forall \alpha \left]0, \alpha_0\right] \qquad \alpha \in \mathbb{R}^{+*}$$

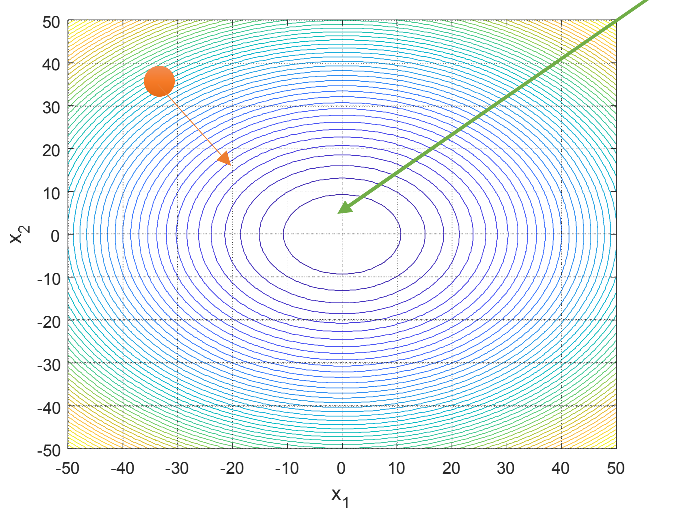
$$d \in \mathbb{R}^{n,1}$$

Minimum recherché

Donc : d donne la direction et α la « taille » du pas



Courbes de niveaux

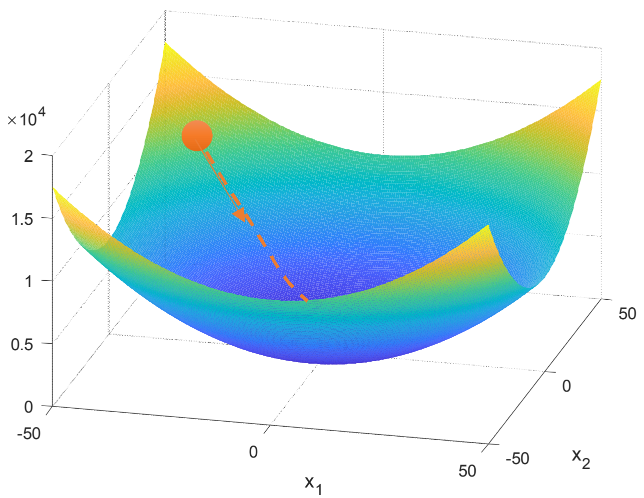


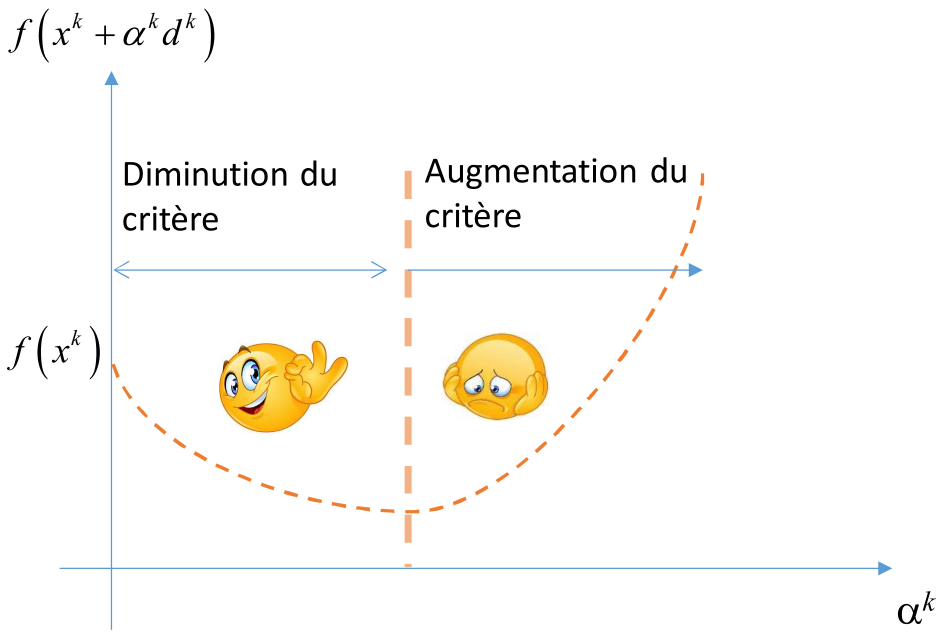
Vocabulaire: direction de descente

Pour une direction de descente d^k:

- Certaines valeurs de pas α^k trop grandes peuvent conduire à $x^{k+1}>x^k$
- Pour une fonction convexe, il existe toujours une valeur de pas α^k qui garantit $x^{k+1}<x^k$ *(tant que x^k n'est pas un minimum local)*

Donc : d donne la direction et α la « taille » du pas





Algorithme générique

Paramètres de l'algorithme:
Solution initiale: x^0

$k = 0$
Tant que *Critère d'arrêt* non satisfait

Calculer une direction de descente d^k

Calculer un pas α^k pour la direction de descente d^k

Mettre à jour la solution : $x^{k+1} = x^k + \alpha^k d^k$

Passage à l'itération suivante: $k = k + 1$

Ce qui différencie les algorithmes:
- Le choix de la direction de descente d^k
- Le choix du pas α^k

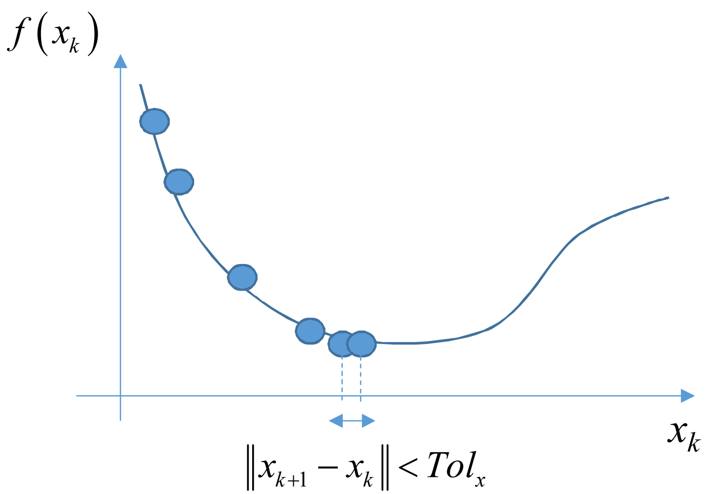
Algorithme générique : Critère d'arrêt

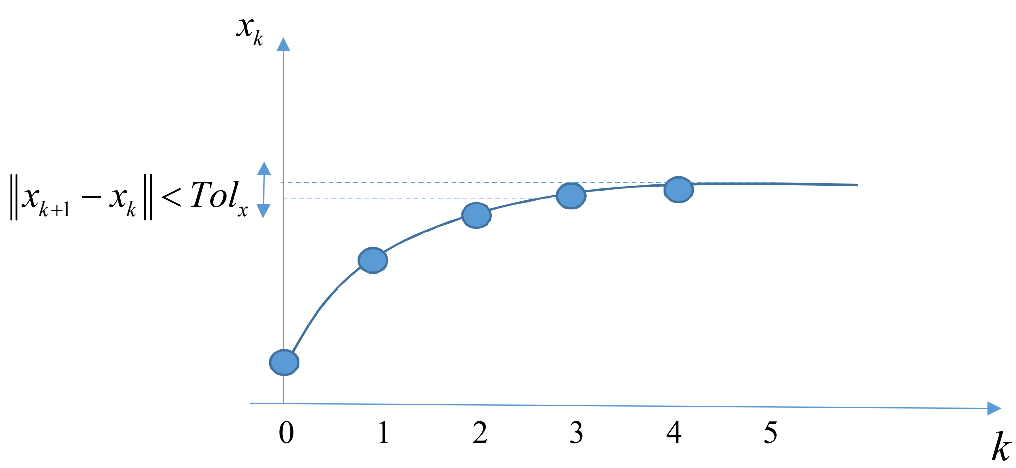
<u>Critères d'arrêts:</u>

En pratique les solveurs permettent de spécifier plusieurs critères. L'algorithme s'arrête dès que l'un d'entre eux est satisfait

- Nombre d'itérations maximal est atteint
- Nombre d'évaluation maximal de la fonction f *atteint*.
- La solution est trouvée avec la bonne précision: $\|x_{k+1} - x_k\| < Tol_x$

  => Permet de maitriser le nombre de décimales significatives





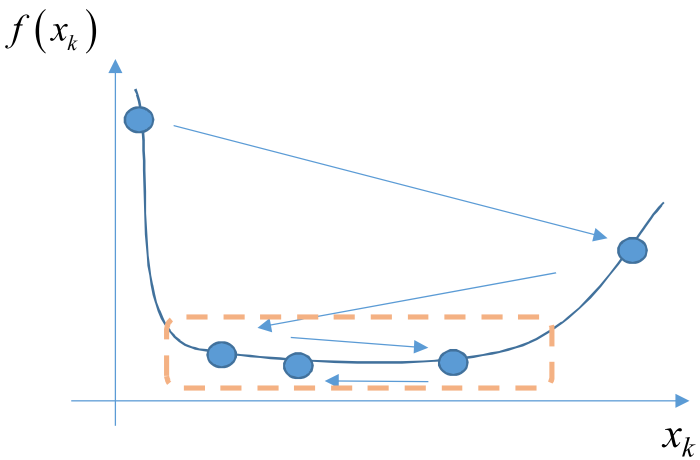
Algorithme générique : Critère d'arrêt

<u>Critères d'arrêts:</u>

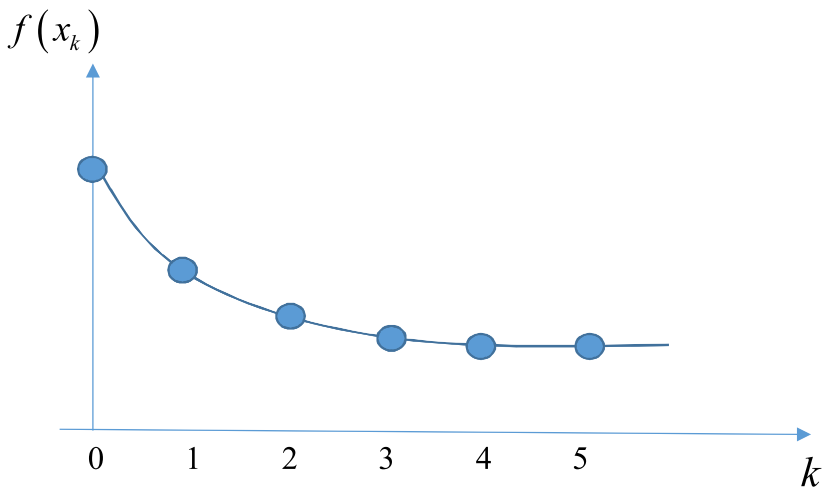
En pratique les solveurs permettent de spécifier plusieurs critères. L'algorithme s'arrête dès que l'un d'entre eux est satisfait

- L'algorithme n'améliore plus la valeur de la fonction: $\left\| f\left(x_{k+1}\right) - f\left(x_k\right) \right\| < Tol_f$

  => Permet d'éviter des itérations inutiles si la fonction f est très « plate » autour du minimum local



Toutes ces valeurs de x_k correspondent quasiment la même valeur du critère



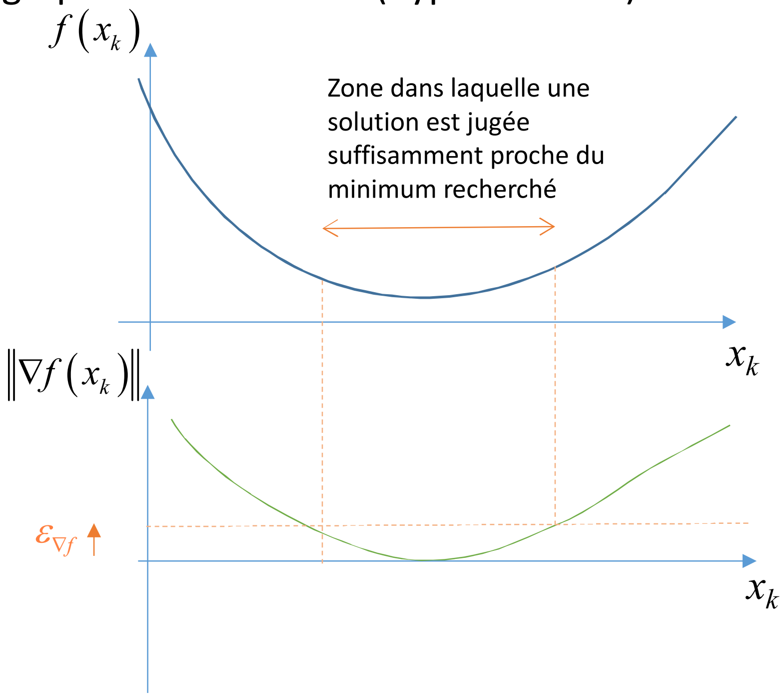
Algorithme générique : Critère d'arrêt

<u>Critères d'arrêts:</u>

En pratique les solveurs permettent de spécifier plusieurs critères. L'algorithme s'arrête dès que l'un d'entre eux est satisfait

- La condition d'optimalité au premier ordre est quasiment vérifiée: $\left\|\nabla f\left(x_k\right)\right\| < \varepsilon_{\nabla f}$

=> Le graphe de la fonction (hyper-surface) est suffisamment « plate »



Algorithme générique : Direction de descente

Paramètres de l'algorithme:

Solution initiale: x^0

$k = 0$

Tant que *Critère d'arrêt* non satisfait

Calculer une direction de descente d^k

Calculer un pas α^k pour la direction de descente d^k

Mettre à jour la solution : $x^{k+1} = x^k + \alpha^k d^k$

Passage à l'itération suivante: $k = k + 1$

Algorithme générique : Direction de descente

On cherche une direction de descente d^k avec $x^{k+1} = x^k + \alpha^k d^k$

On utilise le développement de Taylor en x^k pour une valeur suffisamment petite du pas α^k

$$f\left(x^{k+1}\right) = f\left(x^k + \alpha^k d^k\right) = f\left(x^k\right) + \nabla f\left(x^k\right)^T \alpha^k d^k + O\left(\alpha^k d^k\right)$$

$$f\left(x^{k+1}\right) - f\left(x^k\right) = \nabla f\left(x^k\right)^T \alpha^k d^k + O\left(\alpha^k d^k\right)$$

Pour avoir une direction de descente $f\left(x^{k+1}\right) - f\left(x^k\right) < 0$

$$\nabla f\left(x^k\right)^T \alpha^k d^k + O\left(\alpha^k d^k\right) < 0$$

$$\alpha^k > 0$$

$$\nabla f\left(x^k\right)^T d^k + \frac{O\left(\alpha^k d^k\right)}{\alpha^k} < 0$$

$$\lim_{\alpha^k \to 0^+} \frac{O\left(\alpha^k d^k\right)}{\alpha^k} = 0$$

Finalement, on a une condition pour obtenir une direction de descente :

$$\nabla f\left(x^k\right)^T d^k < 0$$

Direction de descente : Méthode de la plus forte pente

On cherche une direction de descente: $x^{k+1} = x^k + \alpha^k d^k$

Intuition : descendre dans le sens de la plus forte pente: $d^k = -\nabla f\left(x^k\right)$

$$x^{k+1} = x^k - \alpha^k \nabla f\left(x^k\right)$$

C'est évidemment une direction de descente:

$$\nabla f\left(x^k\right)^T d^k = -\nabla f\left(x^k\right)^T \nabla f\left(x^k\right) < 0$$

On montre que c'est la plus forte direction de descente.

Une alternative serait de choisir $d^k = -M \nabla f\left(x^k\right)$ Avec M définie positive

On parle alors de méthode à base de gradient

Direction de descente : Méthode de la plus forte pente

On cherche une direction de descente: $x^{k+1} = x^k + \alpha^k d^k$

Intuition : descendre dans le sens de la plus forte pente: $d^k = -\nabla f\left(x^k\right)$

Problème : la norme du gradient est variable.

Solution : normalisation de la direction (facilitera la recherche de α^k)

$$d^k = -\frac{\nabla f\left(x^k\right)}{\left\|\nabla f\left(x^k\right)\right\|}$$

On obtient toujours la même direction mais la norme du vecteur est toujours unitaire.

Algorithme générique : calcul du pas α^k

Paramètres de l'algorithme:

Solution initiale: x^0

$k = 0$

Tant que *Critère d'arrêt* non satisfait

Calculer une direction de descente d^k

Calculer un pas α^k pour la direction de descente d^k

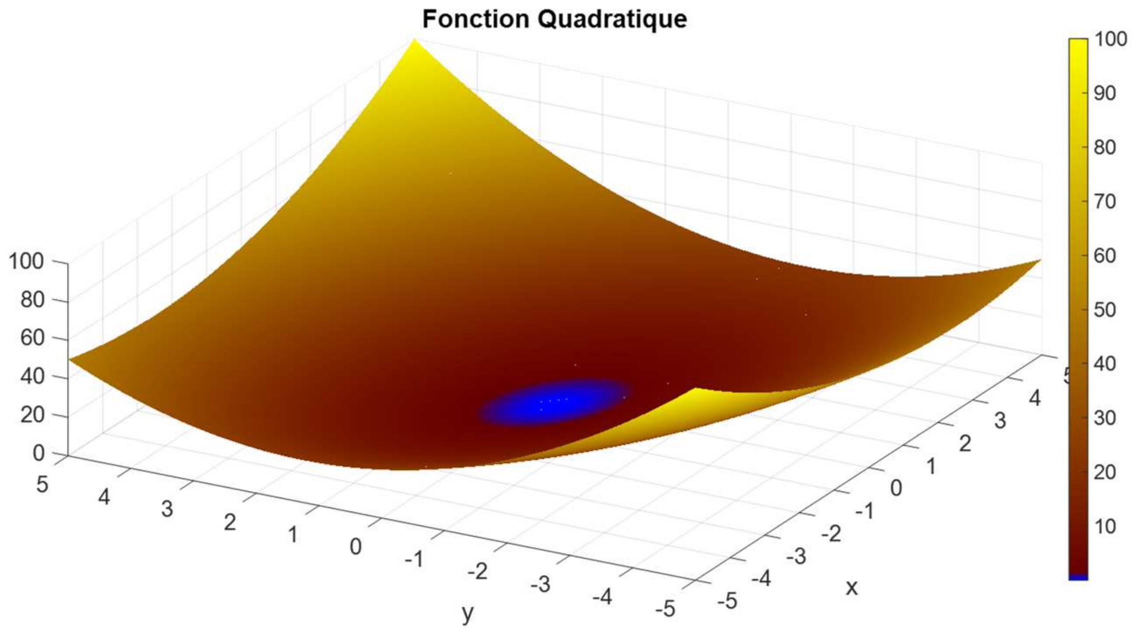
Mettre à jour la solution : $x^{k+1} = x^k + \alpha^k d^k$

Passage à l'itération suivante: $k = k + 1$

Exemple académique : fonction quadratique

$$f(x,y)=\begin{pmatrix}x & y\end{pmatrix}\begin{pmatrix}1 & 0.5\\0.5 & 2\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}$$

```
clear all;
close all;
clc;

H=[1 0.5;
    0.5 2];
f=@(x) [x(1) x(2)]*H*[x(1);x(2)];
xmin=-5;
xmax=5;
ymin=-5;
ymax=5;
Fname='Fonction Quadratique';

% Définition de la colormap
Delta1=(0:10)'/10;
Delta2=(0:750)'/750;
map=[0.4*Delta1 0*Delta1 1-Delta1;
    0.4+0.6*Delta2 1*Delta2 0*Delta2];

% Calcul de la fonction
X=linspace(xmin,xmax,61);
Y=linspace(ymin,ymax,62);
Z=zeros(length(Y),length(X));
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f([X(i);Y(j)]);
    end
end
% Affichage
surf(X,Y,Z,'EdgeColor','none');
xlabel('x');
ylabel('y');
title(Fname)
colorbar;
colormap(map)
hold on;
```



Exemple académique : fonction quadratique

On utilise un pas α constant:

```
clear all;
close all;
clc;
H=[1 0.5;
    0.5 2];

f=@(x) [x(1) x(2)]*H*[x(1);x(2)];
df=@(x) H*[x(1);x(2)]+H'*[x(1);x(2)];
x0=[5;-5];
xmin=-5;
xmax=5;
ymin=-5;
ymax=5;
Fname='Quadratique';

% Définition de la colormap
Delta1=(0:10)'/10;
Delta2=(0:750)'/750;
map=[0.4*Delta1 0*Delta1 1-Delta1;
    0.4+0.6*Delta2 1*Delta2 0*Delta2];

% Criteres d'arret
Tolf=1e-6;
Tolx=1e-9;
IterMax=100;
```

Gradient calculé “manuellement”

```
% Affichage du contour
figure;
X=linspace(xmin,xmax,61);
Y=linspace(ymin,ymax,62);
Z=zeros(length(Y),length(X));
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f([X(i);Y(j)]);
    end
end
surf(X,Y,Z,'EdgeColor','none');
xlabel('x');
ylabel('y');
title(Fname)
colorbar;
colormap(map)
hold on;
```

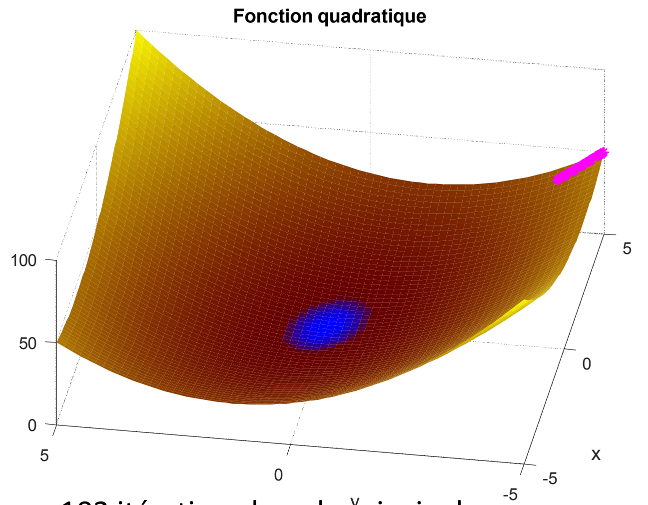
```
% Algorithme
ended=0;
k=1;
fxkp1=f(x0);
x=x0;

while ended==0
    dk=-df(x(:,k));
    dk=dk/norm(dk);
    alpha=0.3;
    x(:,k+1)=x(:,k)+alpha*dk;
    fxk=fxkp1;
    fxkp1=f(x(:,k+1));

    plot3(x(1,k+1),x(2,k+1),fxkp1,'m*');
    plot3([x(1,k+1) x(1,k)],[x(2,k+1) x(2,k)],[fxkp1 fxk],'g');

    ended=(k>IterMax)||...
        norm(fxkp1-fxk)<Tolf||...
        norm(x(:,k)-x(:,k+1))<Tolx;
    k=k+1;
end
fprintf('Valeur finale : x=[%.2f,%.2f] \n',x(1,k),x(2,k))
if k>IterMax
    fprintf(' Iteration maximale atteinte\n');
end
if norm(fxkp1-fxk)<Tolf
    fprintf('Plus d''amélioration de f\n')
end
k=k-1;
if norm(x(:,k)-x(:,k+1))<Tolx
    fprintf('plus d''amélioration de x\n');
end
```

Pas fixe

Exemple académique : fonction quadratique

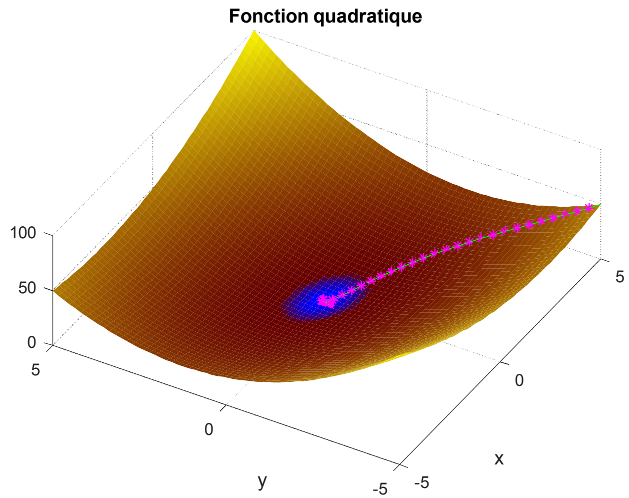
On utilise un pas alpha constant:

$\alpha = 0.01$



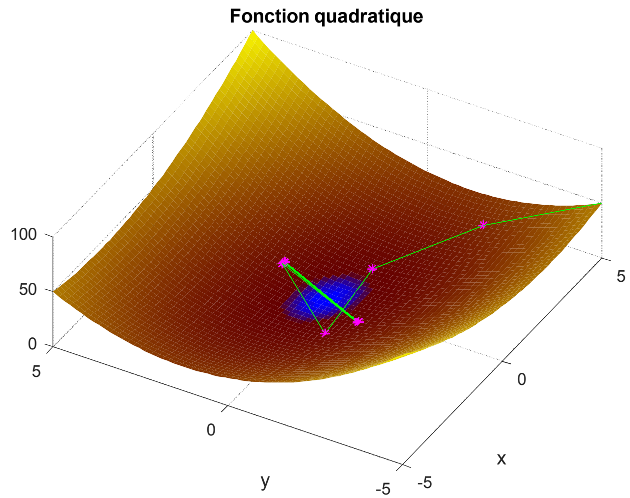
102 itérations boucle principale
102 appels de f

$\alpha = 0.3$



102 itérations boucle principale
102 appels de f

$\alpha = 3$



102 itérations boucle principale
102 appels de f

Problème: la valeur du « pas » α est difficile à choisir, même pour une fonction aussi simple qu'une quadratique.
=> On peut montrer que pour beaucoup d'exemples l'algorithme ne convergera pas ou convergera lentement.

Intuitivement, il faut adapter le pas à la géométrie locale de la fonction.

Algorithme générique : calcul du pas α^k

Le problème à résoudre est le choix du pas α^k qui minimise le cout:

$$\alpha^k = \arg\min_{\alpha>0} f\left(x^k + \alpha d^k\right)$$

Comme $\alpha^k \in \mathbb{R}^{+*}$, le problème est potentiellement « simple » à résoudre (car scalaire).

Plusieurs approches envisageables:

- Résolution exacte: possible que pour des cas particuliers.

En pratique souvent impossible à résoudre pour un problème réaliste.
i.e. convient bien si f est une forme quadratique

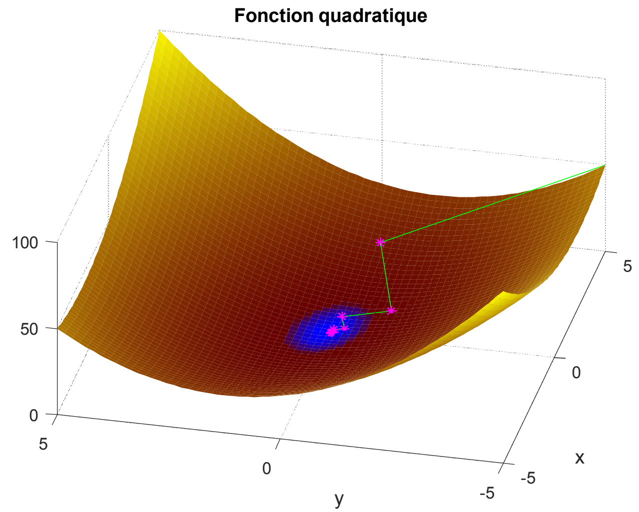
=> Cas d'une fonction quadratique $f(x) = x^T H x + G^T x + A$

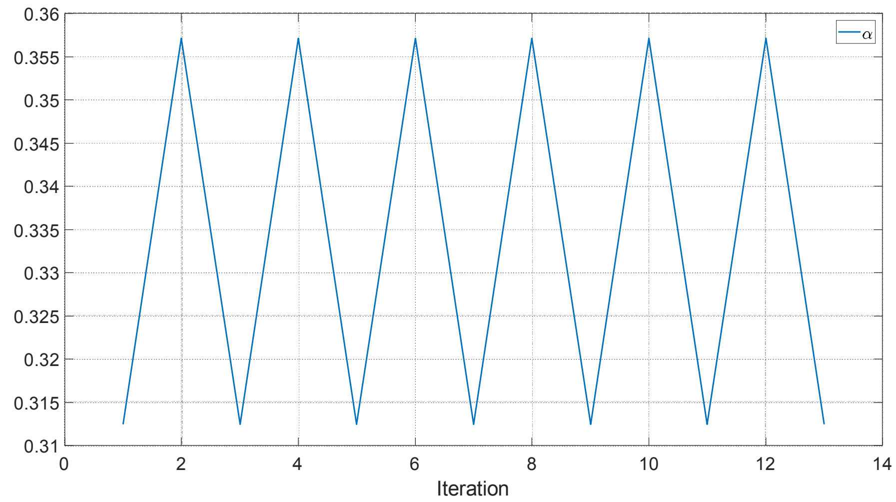
Pour: $$x^{k+1} = x^k - \alpha^k \frac{\nabla f\left(x^k\right)}{\left\|\nabla f\left(x^k\right)\right\|}$$

Le pas optimal est: $$\alpha^k = 0.5 \frac{\left(d^k\right)^T d^k}{\left(d^k\right)^T H d^k} \left\|\nabla f\left(x^k\right)\right\|$$

Exemple académique : fonction quadratique

On utilise un pas $\alpha^k = 0.5 \frac{\left(d^k\right)^T d^k}{\left(d^k\right)^T H d^k} \left\| \nabla f\left(x^k\right) \right\|$ optimal solution de : $\alpha^k = \arg\min_{\alpha > 0} f\left(x^k + \alpha d^k\right)$





Les directions de recherche successives sont alors orthogonales

Problème : en pratique le pas optimal n'est pas calculable.
=> Idée : s'assurer que la valeur de la fonction diminue entre 2 itérations

Algorithme générique : calcul du pas α^k – *Algorithme Backtracking*

Le problème à résoudre est le choix du pas α^k qui minimise le cout: $\alpha^k = \arg\min_{\alpha>0} f\left(x^k + \alpha d^k\right)$

- Algorithme de « Backtracking »

Comme d^k est une direction de descente, il suffit de prendre un pas α assez petit pour avoir

$$f\left(x^k + \alpha d^k\right) < f\left(x^k\right)$$

Idée : partir d'une pas initial et le diminuer jusqu'à ce que la condition soit satisfaite.

<u>Algorithme au pas k :</u>

Paramètres : $\alpha_0^k > 0, \beta \in]0,1[$

$i = 0$

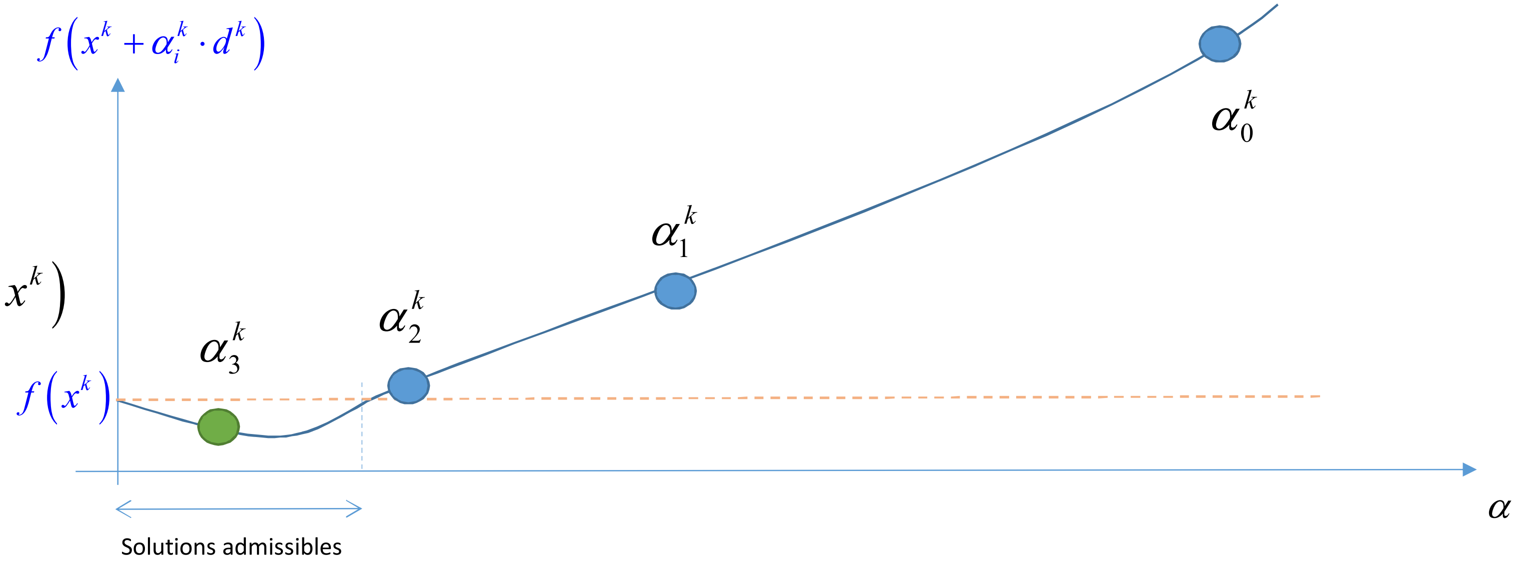
Tant que $f\left(x^k + \alpha_i^k d^k\right) \geq f\left(x^k\right)$

$\alpha_{i+1}^k = \alpha_i^k \cdot \beta$

$i = i + 1$

Fin Tant que

$\alpha^k = \alpha_i^k$



Algorithme générique : calcul du pas α^k – *Algorithme Backtracking*

Modifications du programme précédent:

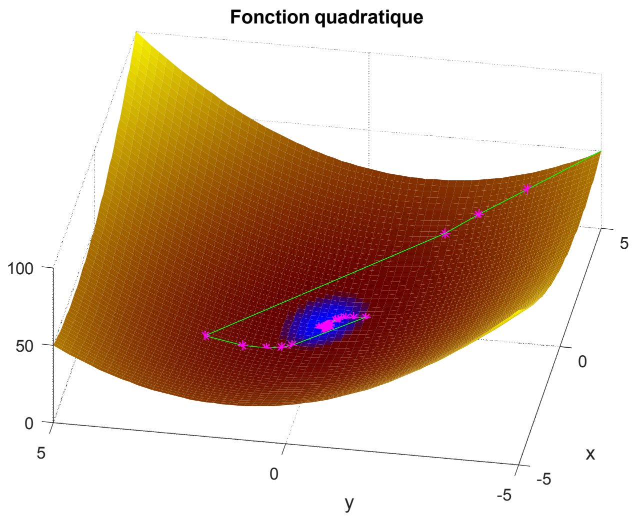
```
% Paramtères backtracking
alpha0=1;
beta=0.9;

while ended==0
    dk=-df(x(:,k));

    alpha(k)=BackTracking(f,x(:,k),dk,alpha0,beta);

    x(:,k+1)=x(:,k)+alpha(k)*dk;
    fxk=fxkp1;
    fxkp1=f(x(:,k+1));

    plot3(x(1,k+1),x(2,k+1),fxkp1,'m*');
    plot3([x(1,k+1) x(1,k)],[x(2,k+1) x(2,k)],[fxkp1 fxk],'g');

    ended=(k>IterMax)||...
        norm(fxkp1-fxk)<Tolf||...
        norm(x(:,k)-x(:,k+1))<Tolx;
    k=k+1;
end


function alpha=BackTracking(ff,xk,dk,alpha0,beta)
fxk=f(xk);
alpha=alpha0;
while fxk<f(xk+alpha*dk)
   alpha=alpha*beta;
end
end
```

NB: il est possible d’optimiser le programme pour économiser des appels à la fonction f…

Exemple académique : fonction quadratique

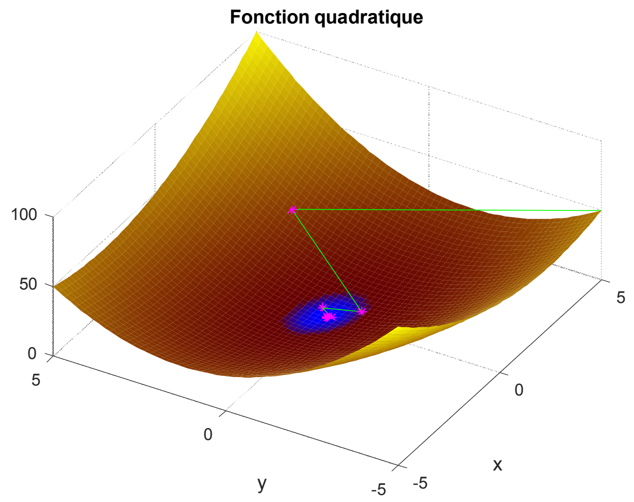
On utilise un pas calculé avec l'algorithme de Backtracking avec: $\alpha_0^k = 1$

$\beta = 0.1$



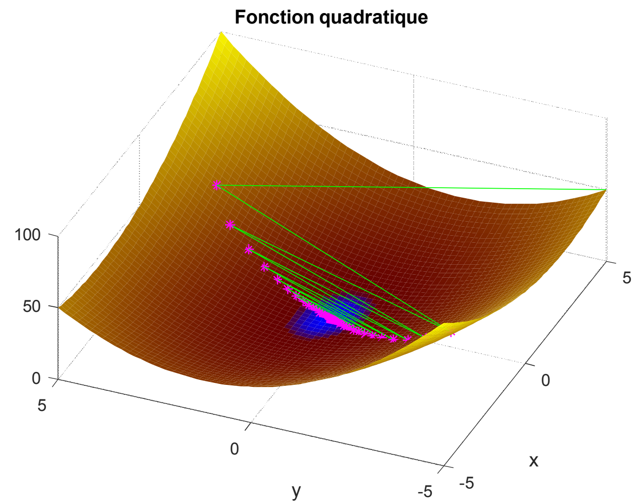
37 itérations boucle principale
137 appels de f

$\beta = 0.5$



12 itérations boucle principale
49 appels de f

$\beta = 0.9$



79 itérations boucle principale
855 appels de f

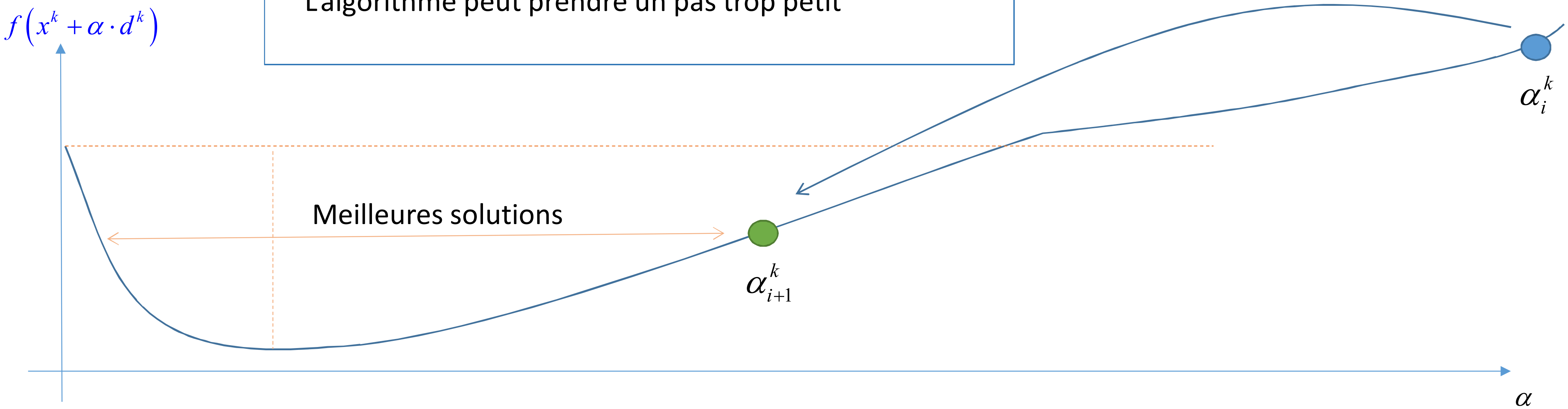
Problème: l'algorithme de backtracking s'arrête à la première amélioration de f
=> Peut être que des pas plus petits seraient plus efficace.

Algorithme générique : calcul du pas α^k – *Algorithme Backtracking*

Le problème à résoudre est le choix du pas α^k qui minimise le cout: $\alpha^k = \arg\min_{\alpha>0} f\left(x^k + \alpha d^k\right)$

- **<u>Algorithme de « Backtracking »</u>**

Avantage :
L'algorithme garanti qu'un pas sera trouvé
Inconvénient :
L'algorithme ne permet pas de diminuer f suffisamment
L'algorithme peut prendre un pas trop petit

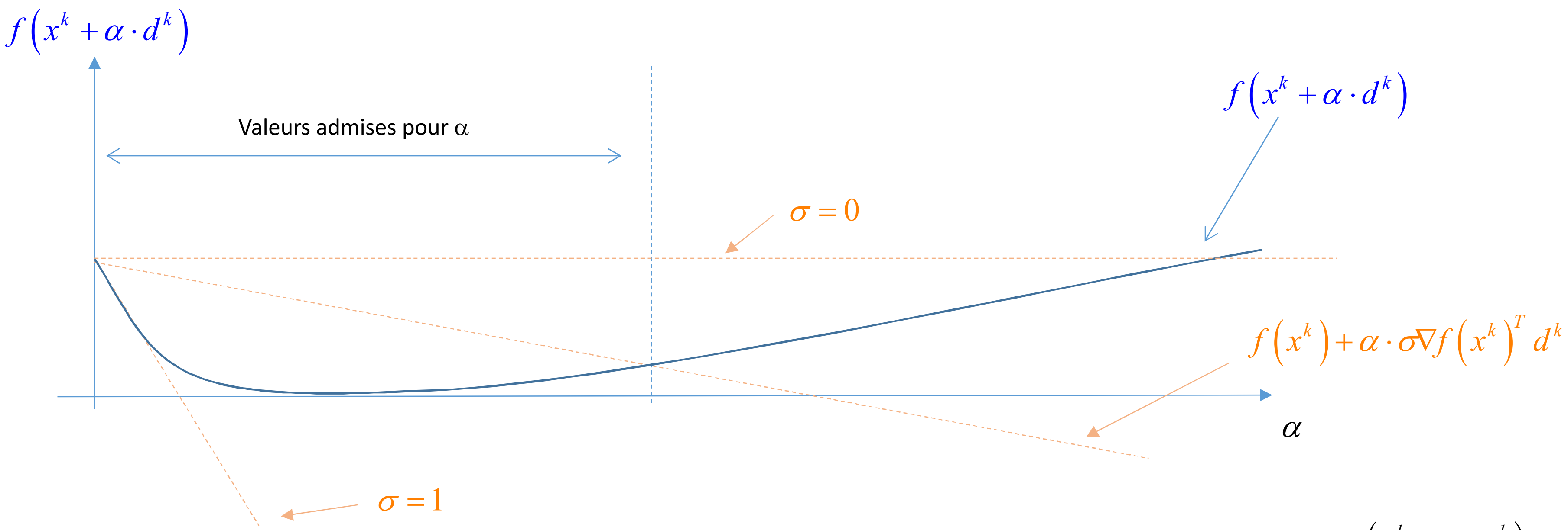


Algorithme générique : calcul du pas α^k – *Règle d'Armijo*

- **Règle d'Armijo:**

/Armijo 1966/

Idée : on cherche α tel que $f\left(x^k+\alpha\cdot d^k\right)\leq \boxed{f\left(x^k\right)+\alpha\cdot\sigma\cdot\nabla f\left(x^k\right)^T d^k}$ Eq. d'une droite avec une pente négative

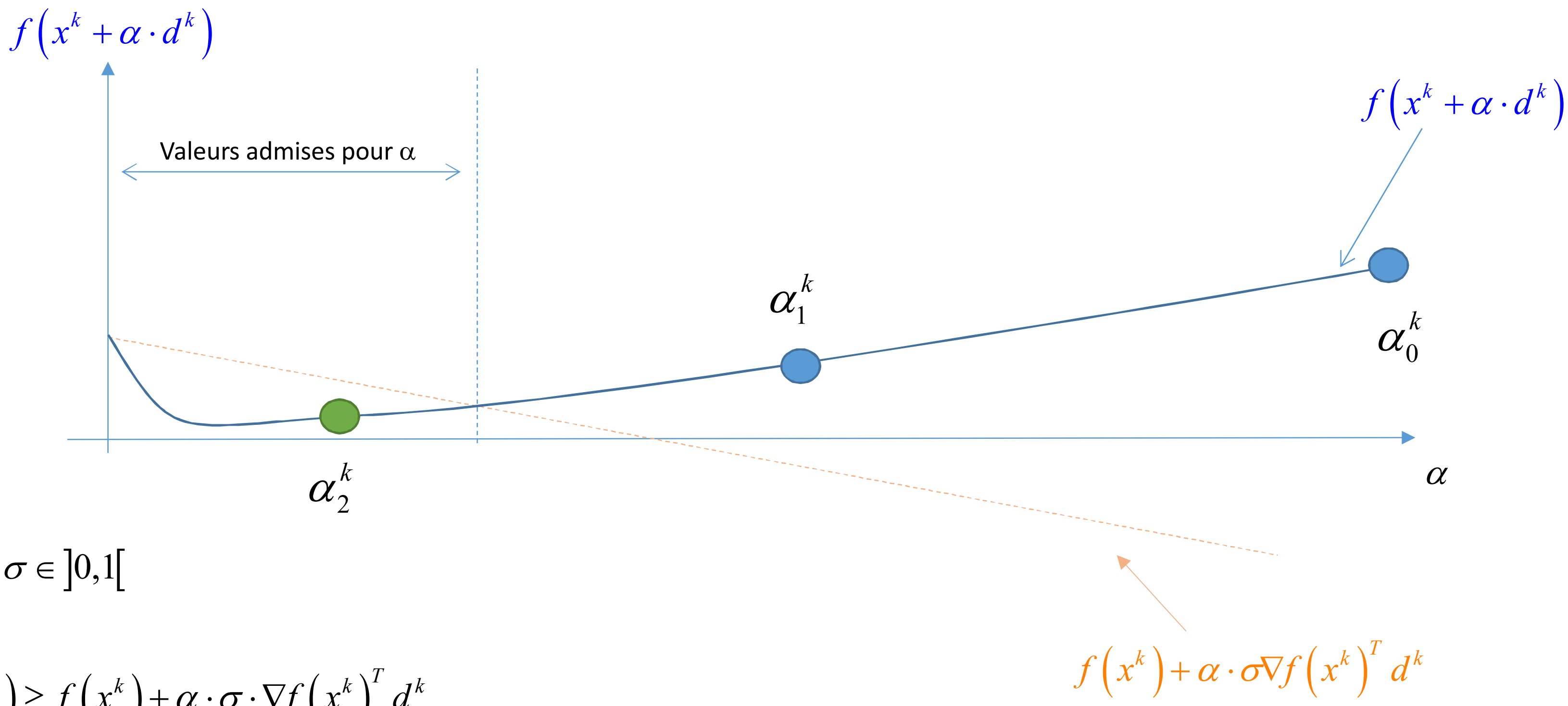


Le paramètre $\sigma\in]0,0.5[$ permet de choisir la pente de la droite, donc la diminution de $f\left(x^k+\alpha\cdot d^k\right)-f\left(x^k\right)$:

- $\sigma\rightarrow 0^+$ Algorithme de *backtracking* mais diminution de $f(x^{k+1})$ potentiellement faible
- $\sigma\rightarrow 1^-$ Diminution maximale, en général inatteignable car la courbe est souvent au dessus de la tangente
- $\sigma\in]0,1[$ Potentiellement « grande » diminution de $f(x^{k+1})$ mais requiert plus de calculs
- $\sigma\in]0,0.5[$ En pratique, $\sigma<0.5$ permet de garantir la convergence pour une fonction f quadratique

Algorithme générique : calcul du pas α^k – *Règle d'Armijo*

- **<u>Règle d'Armijo:</u>**



<u>Algorithme au pas k :</u>

Paramètres : $\alpha_0^k > 0, \sigma \in]0,1[$

$i = 0$

Tant que $f\left(x^k+\alpha\cdot d^k\right) \geq f\left(x^k\right)+\alpha\cdot\sigma\cdot\nabla f\left(x^k\right)^T d^k$

$\alpha_{i+1}^k = \alpha_i^k \cdot \beta$

$i = i+1$

Fin Tant que

$\alpha^k = \alpha_i^k$

<u>Avantage:</u> garantie de la décroissance
<u>Inconvénient :</u> le pas peux devenir trop petit

Algorithme générique : calcul du pas α^k – *Algorithme Backtracking*

Modifications du programme précédent:

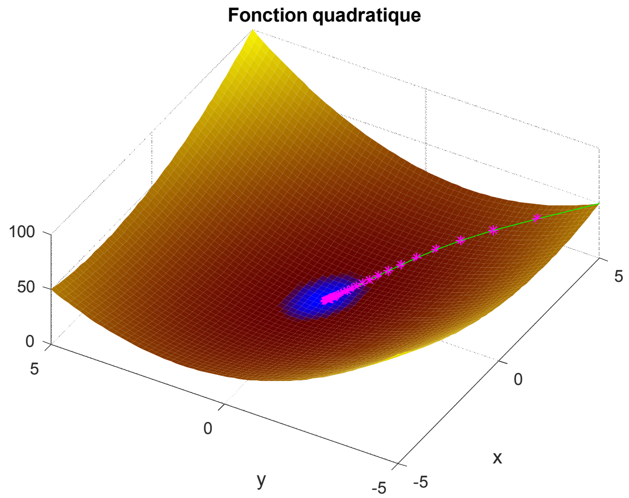
```
% Paramètres Armijo
alpha0=1;
beta=0.9;
gamma=0.5;

while ended==0
    dk=-df(x(:,k));

    [alpha(k)]=Armijo(ff,x(:,k),dk,-dk,alpha0,beta,gamma);

    x(:,k+1)=x(:,k)+alpha(k)*dk;
    fxk=fxkp1;
    fxkp1=f(x(:,k+1));

    plot3(x(1,k+1),x(2,k+1),fxkp1,'m*');
    plot3([x(1,k+1) x(1,k)],[x(2,k+1) x(2,k)],[fxkp1 fxk],'g');

    ended=(k>IterMax)||...
        norm(fxkp1-fxk)<Tolf||...
        norm(x(:,k)-x(:,k+1))<Tolx;
    k=k+1;
end


function alpha=Armijo(f,xk,dk,GradF,alpha0,beta,gamma)
alpha=alpha0;
fxk=f(xk);
while f(xk+alpha*dk)>=fxk+alpha*gamma*GradF'*dk
    alpha=alpha*beta;
end
end
```

NB: il est possible d'optimiser le programme pour économiser des appels à la fonction f…

Exemple académique : fonction quadratique

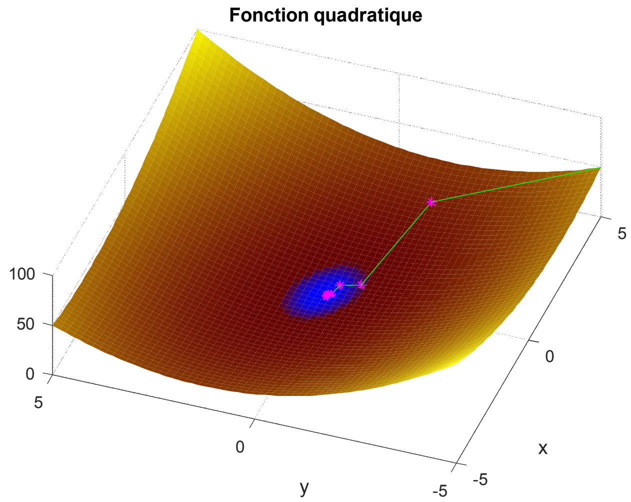
On utilise un pas calculé avec l'algorithme d'Armijo avec: $\alpha_0^k = 1$ $\sigma = 0.5$

$\beta = 0.1$



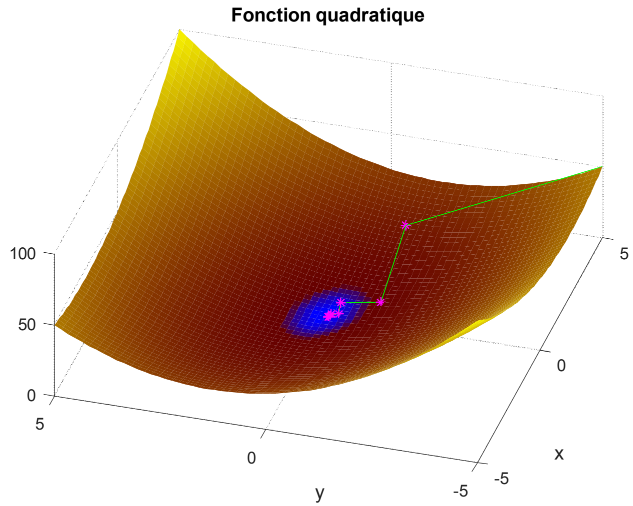
49 itérations boucle principale
192 appels de f

$\beta = 0.5$



11 itérations boucle principale
45 appels de f

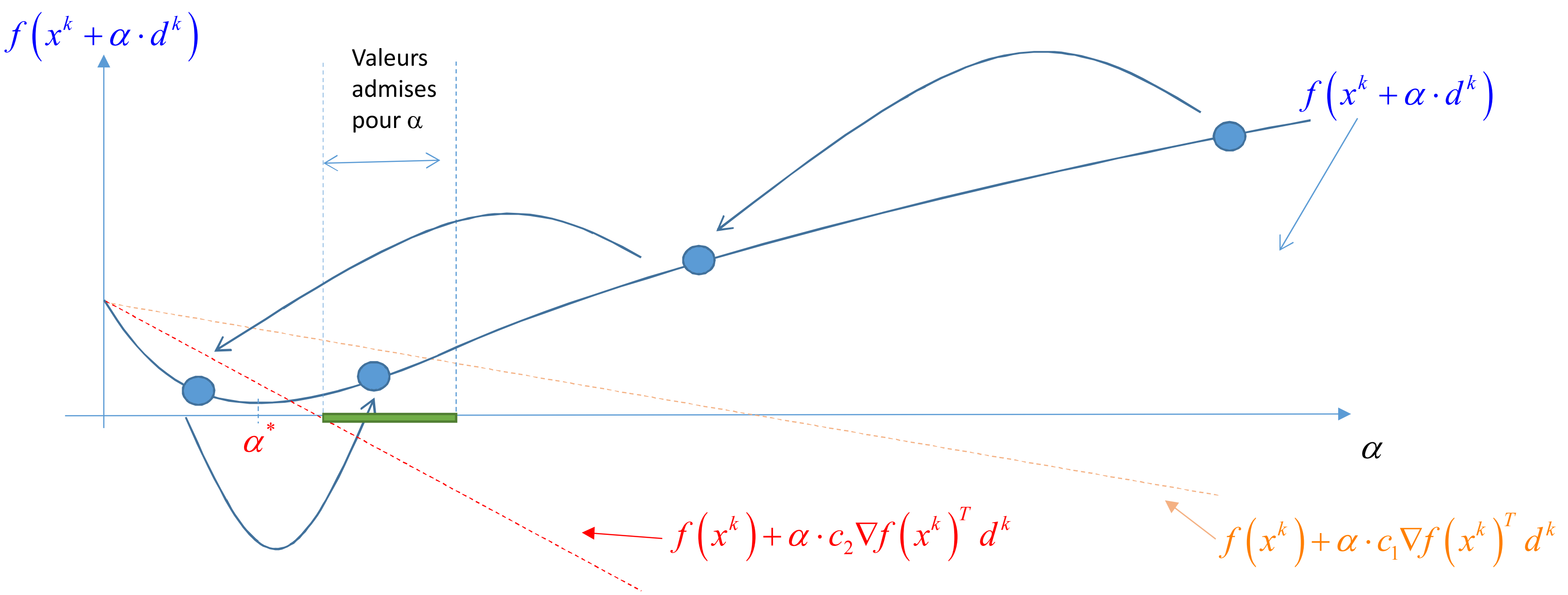
$\beta = 0.9$



12 itérations boucle principale
143 appels de f

Problème: l'algorithme d'Armijo garanti que la fonction f décroit mais rien ne dit que le pas choisit est proche d'un optimal
Idée : utiliser les conditions d'optimalité au premier ordre

Algorithme générique : calcul du pas α^k – *Armijo-Goldstein*



Règles d'Armijo-Goldstein

Le pas α doit vérifier:

$$f\left(x^k + \alpha \cdot d^k\right) \leq f\left(x^k\right) + \alpha \cdot c_1 \cdot \nabla f\left(x^k\right)^T d^k$$

$$f\left(x^k + \alpha \cdot d^k\right) \geq f\left(x^k\right) + \alpha \cdot c_2 \cdot \nabla f\left(x^k\right)^T d^k$$

$$0 < c_1 < c_2 < 1$$

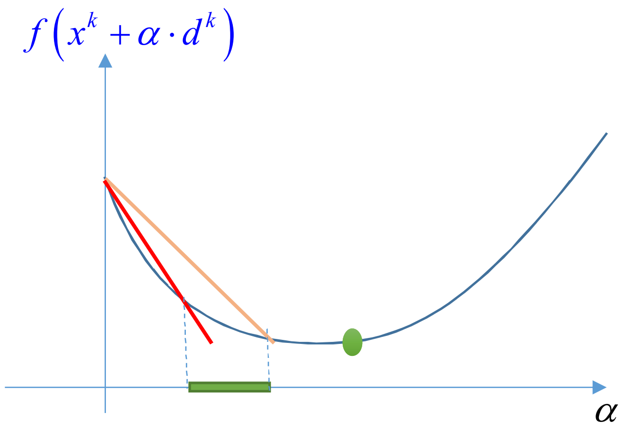
Mise en oeuvre: Ne pas accepter les pas trop petits

Algorithme générique : calcul du pas α^k – *Armijo-Goldstein*

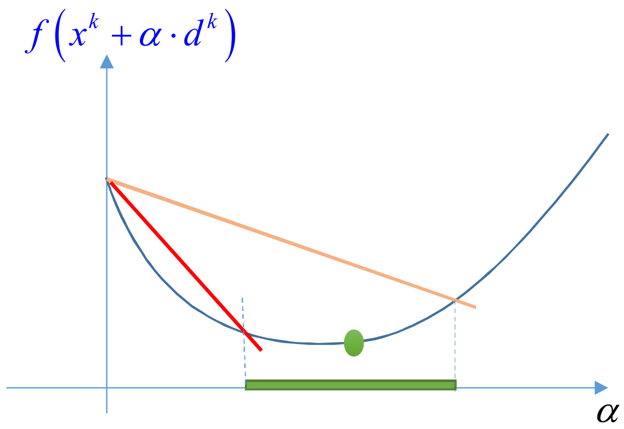
```
function alpha=ArmijoGoldstein(f,xk,dk,GradF,alpha0,c1,c2,beta)
alpha=alpha0;
fxk=f(xk);
while (f(xk+alpha*dk)>fxk+alpha*c1*GradF'*dk)||...
        (f(xk+alpha*dk)<fxk+alpha*c2*GradF'*dk)

    if f(xk+alpha*dk)>=fxk+alpha*c1*GradF'*dk
        alpha=alpha*beta;
    else
        alpha=alpha/beta; % Annule le dernier pas
        beta=beta*0.5;
    end
end

end
```

<u>Problème lié au réglage des paramètres:</u>

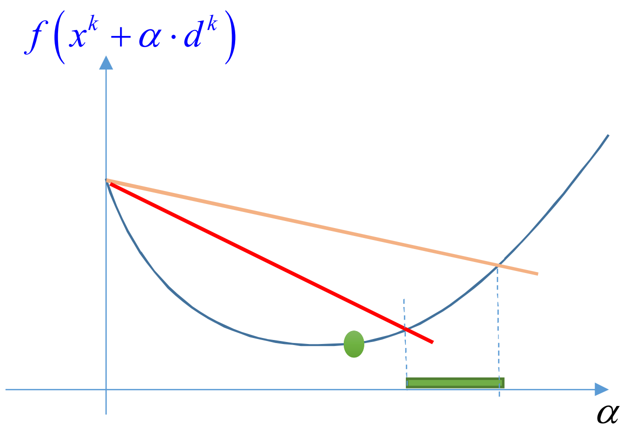
- Suivant les valeurs de c_1 et c_2, le pas α peut être trop long ou trop cout
- Comment trouver un bon réglage?



Trop court



Bon réglage

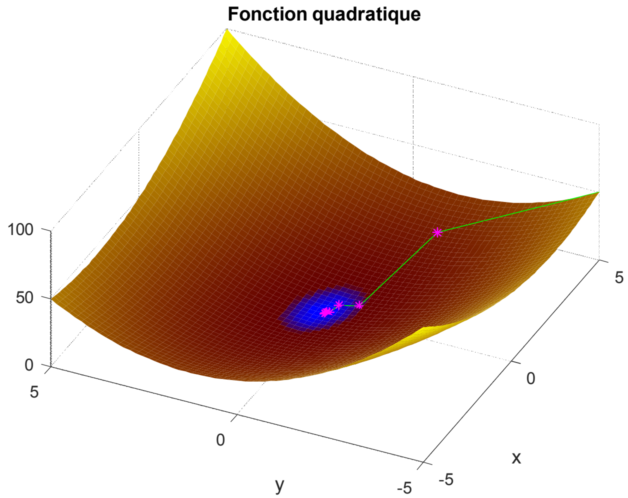


Trop long

Exemple académique : fonction quadratique

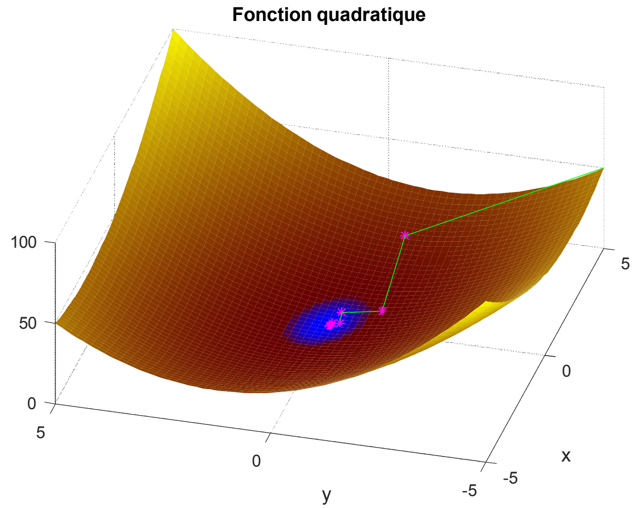
On utilise un pas calculé avec l'algorithme d'Armijo-Goldstein avec: $\alpha_0^k = 1$ $c_1 = 0.5$ $c_2 = 0.1$

$\beta = 0.5$



11 itérations boucle principale
60 appels de f

$\beta = 0.9$



9 itérations boucle principale
82 appels de f

Nb: ne change pas grand-chose sur cet exemple car il n'y avait pas de pb de petits pas

Algorithme générique : calcul du pas α^k – *Règle de Wolfe*

Amélioration :

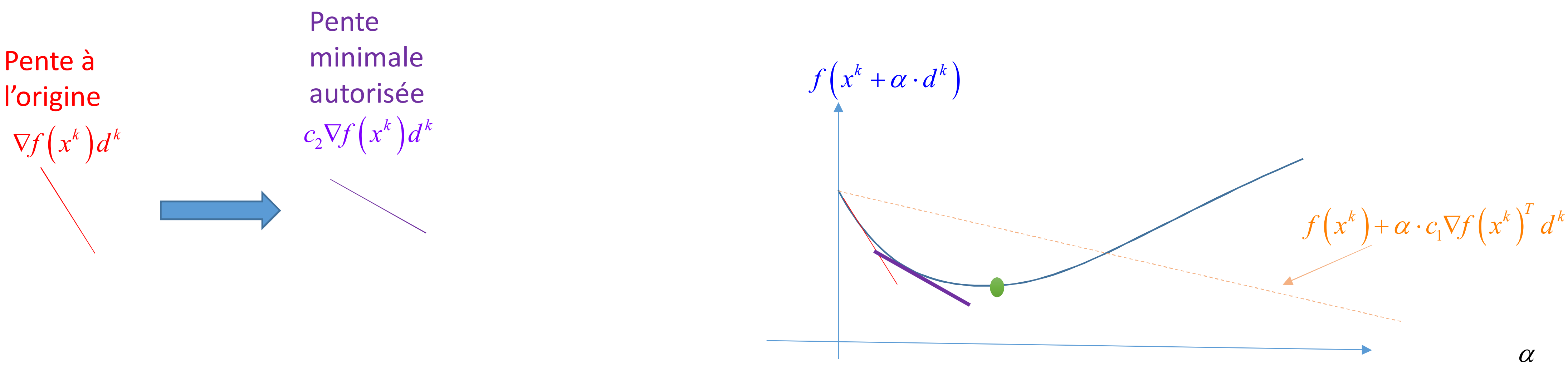
- Tenir compte de la condition d'optimalité au premier ordre : $\dfrac{df\left(x^k+\alpha\cdot d^k\right)}{d\alpha}=\nabla f\left(x^k+\alpha\cdot d^k\right)^T\cdot d^k=0$

Problème :

- La résolution exacte est trop compliquée => utilisation d'une solution « approché »

Idée :

- Garantir que la borne min permet de rester à gauche de la solution recherchée



Algorithme générique : calcul du pas α^k – *Règle de Wolfe*

<u>Règles de wolfe:</u> $0 < c_1 < c_2 < c1$

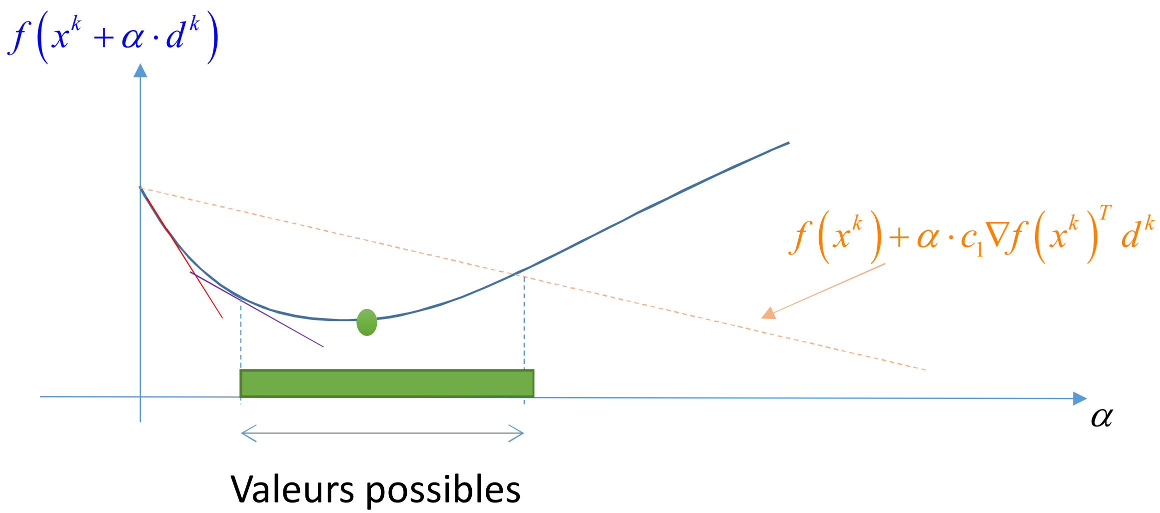
1) La fonction doit décroitre suffisamment (Armijo):

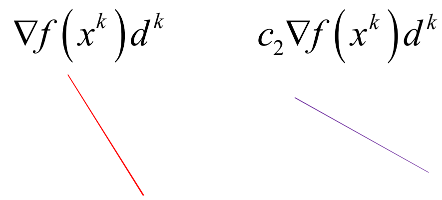
$$f\left(x^k + \alpha \cdot d^k\right) \leq f\left(x^k\right) + \alpha \cdot c_1 \cdot \nabla f\left(x^k\right)^T d^k$$

Valeurs typiques : $c_1 = 10^{-4}$ $c_2 = 0{,}1\text{-}0{,}9$

2) La pente doit être « moins négative » qu'à l'origine

$$\nabla f\left(x^k + \alpha \cdot d^k\right)^T d^k \geq c_2 \cdot \nabla f\left(x^k\right)^T d^k$$





Algorithme générique : calcul du pas α^k – *Règle de Wolfe*

Règles de wolfe:

$$f\left(x^k + \alpha \cdot d^k\right) \leq f\left(x^k\right) + \alpha \cdot c_1 \cdot \nabla f\left(x^k\right)^T d^k$$
$$\nabla f\left(x^k + \alpha \cdot d^k\right)^T d^k \geq c_2 \cdot \nabla f\left(x^k\right)^T d^k$$

Valeurs typiques : $c_1=10^{-4}$ $c_2=0{,}1\text{-}0{,}9$

$\Rightarrow$ La mise en œuvre est assez difficile
$\Rightarrow$ Trouver un code existant :

Par exemple « *Optimization tutorial* » de Mark Bangert sur Matlab Central

Un autre code : http://www4.ncsu.edu/~kksivara/ma706

Un autre code : https://github.com/GuipengLi

Algorithme générique : calcul du pas α^k – *Règle de Wolfe*

Algorithme proposé par /Jorge Nocedal & Stephen J. Wright, Numerical Optimization/

$$f(x_k + \alpha_k p_k) \le f(x_k) + c_1 \alpha_k \nabla f_k^T p_k, \tag{3.6a}$$

$$\nabla f(x_k + \alpha_k p_k)^T p_k \ge c_2 \nabla f_k^T p_k, \tag{3.6b}$$

Algorithm 3.5 (Line Search Algorithm).

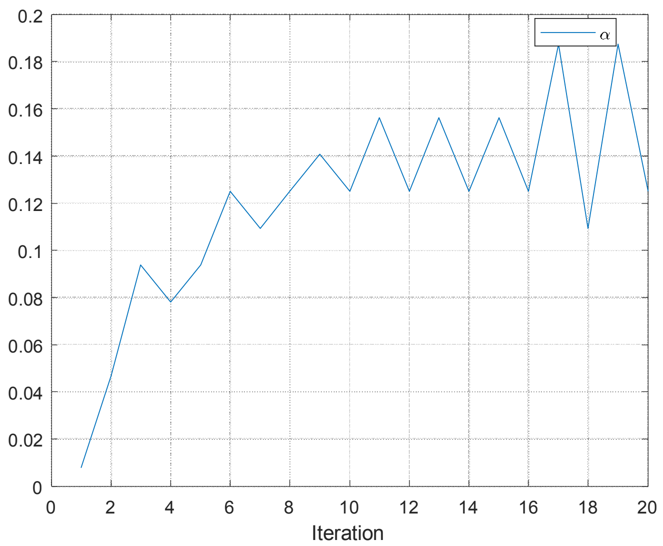
Set $\alpha_0 \leftarrow 0$, choose $\alpha_{\max} > 0$ and $\alpha_1 \in (0, \alpha_{\max})$;
$i \leftarrow 1$;
repeat
    Evaluate $\phi(\alpha_i)$;
    if $\phi(\alpha_i) > \phi(0) + c_1 \alpha_i \phi'(0)$ or $[\phi(\alpha_i) \ge \phi(\alpha_{i-1})$ and $i > 1]$
        $\alpha_* \leftarrow$**zoom**(α_{i-1}, α_i) and **stop**;
    Evaluate $\phi'(\alpha_i)$;
    if $|\phi'(\alpha_i)| \le -c_2 \phi'(0)$
        set $\alpha_* \leftarrow \alpha_i$ and **stop**;
    if $\phi'(\alpha_i) \ge 0$
        set $\alpha_* \leftarrow$**zoom**(α_i, α_{i-1}) and **stop**;
    Choose $\alpha_{i+1} \in (\alpha_i, \alpha_{\max})$;
    $i \leftarrow i + 1$;
end (repeat)

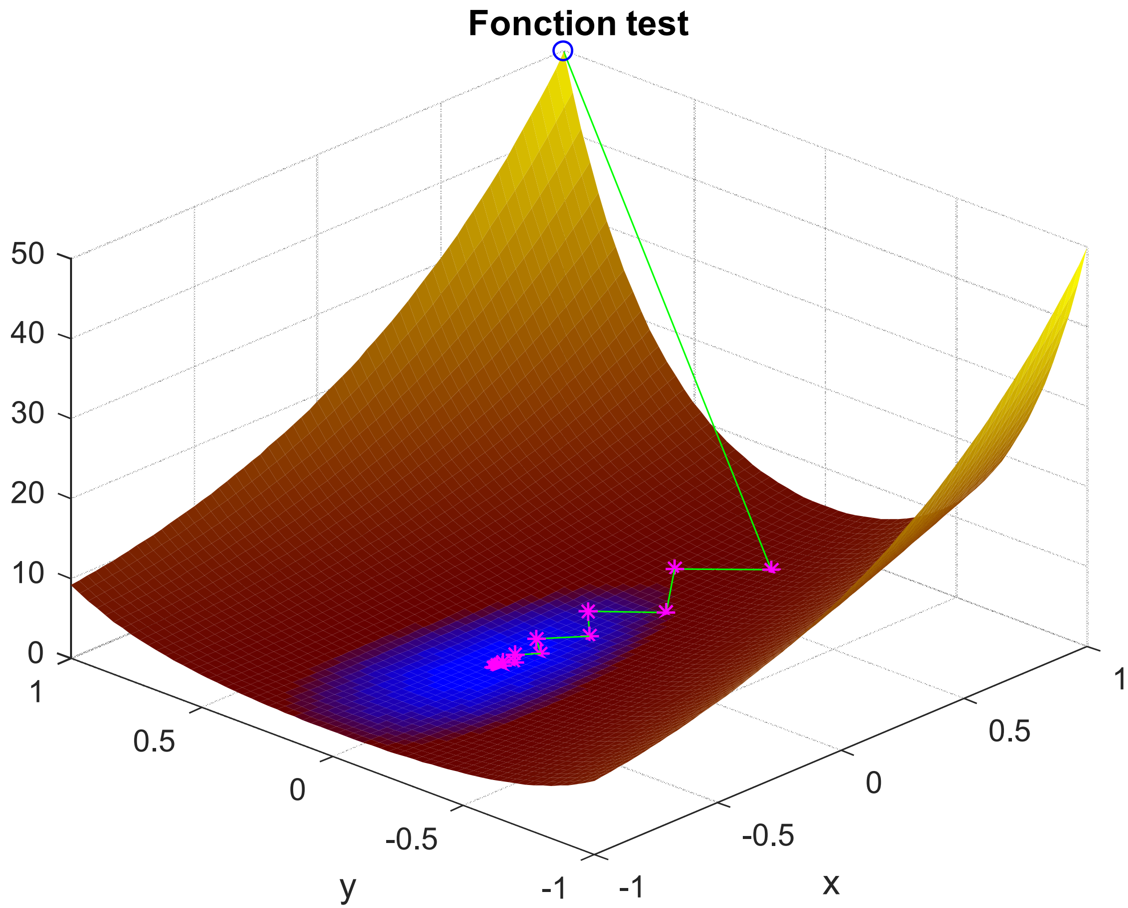
Algorithm 3.6 (zoom).

repeat
    Interpolate (using quadratic, cubic, or bisection) to find
        a trial step length α_j between α_{lo} and α_{hi};
    Evaluate $\phi(\alpha_j)$;
    if $\phi(\alpha_j) > \phi(0) + c_1 \alpha_j \phi'(0)$ or $\phi(\alpha_j) \ge \phi(\alpha_{lo})$
        $\alpha_{hi} \leftarrow \alpha_j$;
    else
        Evaluate $\phi'(\alpha_j)$;
        if $|\phi'(\alpha_j)| \le -c_2 \phi'(0)$
            Set $\alpha_* \leftarrow \alpha_j$ and **stop**;
        if $\phi'(\alpha_j)(\alpha_{hi} - \alpha_{lo}) \ge 0$
            $\alpha_{hi} \leftarrow \alpha_{lo}$;
        $\alpha_{lo} \leftarrow \alpha_j$;
end (repeat)

Exemple académique : fonction quadratique

On utilise un pas calculé avec l'algorithme de Wolfe: $\alpha_0^k = 1$ $c_1 = 10^{-4}$ $c_2 = 0.1$





21 itérations boucle principale
151 appels de f

1) Contexte
2) Algorithme à base de Gradient
3) Méthode de Newton & quasi-Newton
4) Méthode « Régions de confiance » (Trust region)

Méthode de Newton

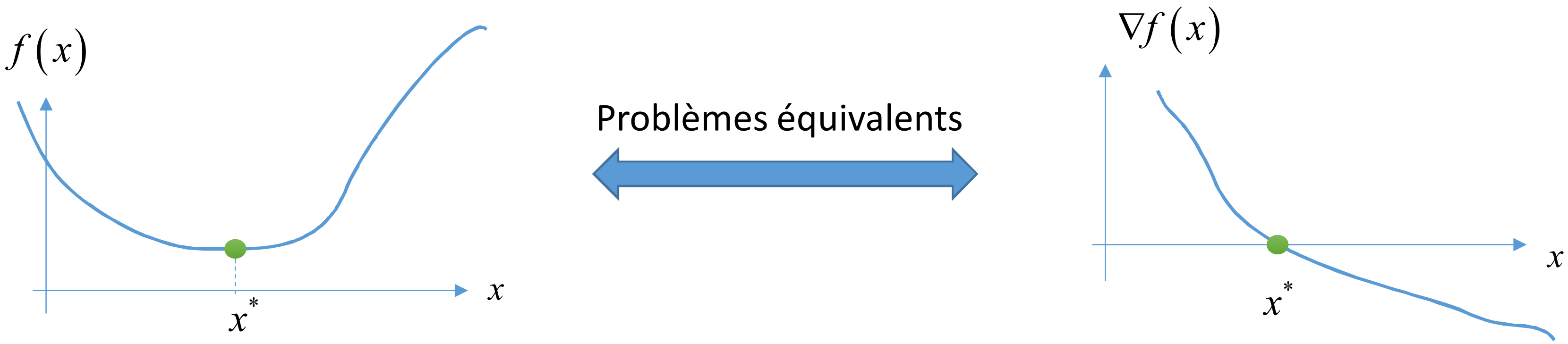
Les algorithmes précédents n'exploitent qu'une information à l'ordre 1.

=> L'algorithme progresse dans une direction qui dépends du gradient

Pour avancer plus vite, une approche consiste à utiliser la condition d'optimalité :

$$\nabla f\left(x^*\right) = 0$$

=> Le problème de recherche du minimum est transformé en la recherche d'un zéro d'une fonction multivariables.



Méthode de Newton

Rappel : Méthode de Newton pour la recherche du zéro d'une fonction

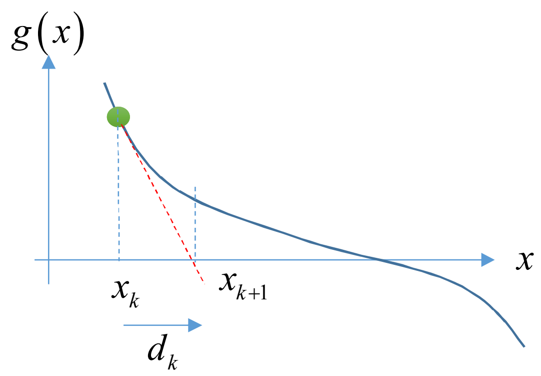
$$g\left(x^k + d^k\right) \approx g\left(x^k\right) + \nabla g\left(x^k\right) d^k$$

$$g\left(x^k + d^k\right) = 0 \Leftrightarrow -g\left(x^k\right) \approx \nabla g\left(x^k\right) d^k$$

D'où la méthode de Newton

Résoudre: $-g\left(x^k\right) = \nabla g\left(x^k\right) d^k$

Itérer $x^{k+1} = x^k + d^k$



Condition d'optimalité au premier ordre pour $\min_{x \in \mathbb{R}^n} f(x)$

$$\nabla f\left(x^*\right) = 0$$

Idée: Appliquer la méthode qui recherche les zéros du gradient $\nabla f\left(x^*\right) = 0$

On pose $g(x) = \nabla f(x)$ d'où $\begin{cases} -\nabla f\left(x^k\right) \approx \nabla^2 f\left(x^k\right) d^k \\ x^{k+1} = x^k + d^k \end{cases}$

Méthode de Newton

Algorithme de base:

Paramètres :
- x^0 : solution initiale
- ε : tolérence

Hessien

$k = 0$

Répéter

Chercher d^k tel que $\boxed{-\nabla f\left(x^k\right) \approx \nabla^2 f\left(x^k\right) d^k}$

$x^{k+1} = x^k + d^k$

$k = k + 1$

Tant que $\left\|\nabla f\left(x^k\right)\right\| < \varepsilon$

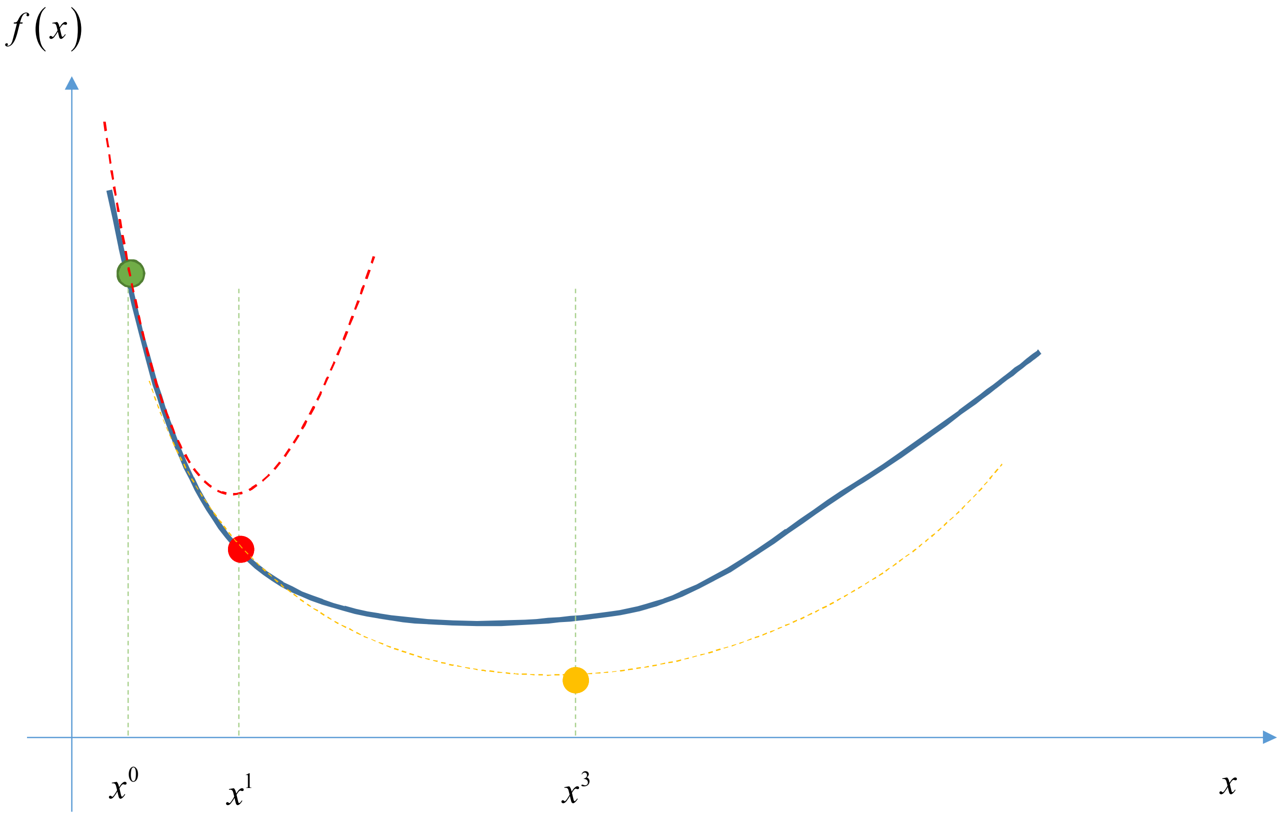
Système linéaire en d^k

Les problèmes à surmonter:
- Résolution de $-\nabla f\left(x^k\right) \approx \nabla^2 f\left(x^k\right) d^k$

  Le Hessien n'est pas nécessairement inversible
- Pas de distinction entre minimum ou maximum local
- Le Hessien n'est pas forcément connu

Méthode de Newton

En fait l'algorithme revient à faire une approximation à l'ordre 2 de f et à chercher itérativement le minimum.

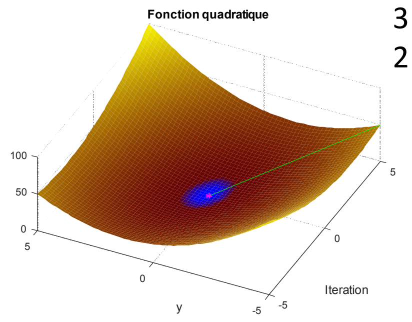


Exemple académique : fonction quadratique

Sur un exemple quadratique, la méthode fonctionne parfaitement (aux erreurs numériques près)

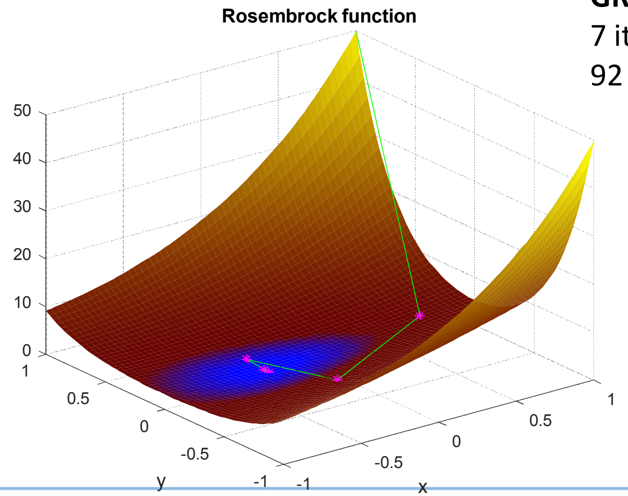
=> Nécessité de comparer sur un exemple plus complexe

En général, fonctionne mal car des problèmes numériques surviennent lors de la résolution du système linéaire



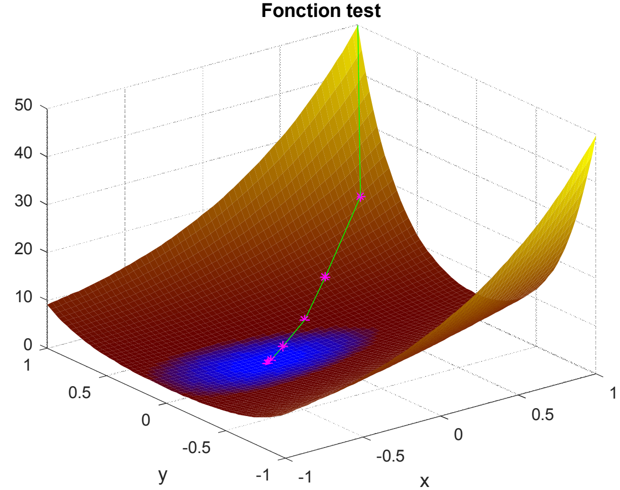
3 itérations boucle principale
2 appels de f

$$f(x,y)=e^{x-3y-\frac{1}{10}}+e^{x+3y-\frac{1}{10}}+e^{-x-\frac{1}{10}}$$



GRADIENT+ARMIJO
7 itérations boucle principale
92 appels de f

NEWTON
9 itérations boucle principale
9 appels de f



Méthode de quasi-Newton

Idée:

Si le Hessien est inversible $$-\nabla f\left(x^k\right) \approx \nabla^2 f\left(x^k\right) d^k \Leftrightarrow d^k = -\left(\nabla^2 f\left(x^k\right)\right)^{-1} \cdot \nabla f\left(x^k\right)$$
$$x^{k+1} = x^k + d^k$$

On retrouve une forme « similaire » à celle de l'algorithme à base de gradient

d^k est une direction de descente si et seulement si le Hessien est définit positif

On remplace l'inverse du Hessien, souvent inconnue, par une approximation:

$$d^k = -\alpha^k S^k \cdot \nabla f\left(x^k\right)$$
$$x^{k+1} = x^k + d^k$$

=> Les algorithmes doivent « rendre » l'approximation du Hessien positive

Différentes approximations donnent différentes méthodes:

- Davidson-Fletcher-Powell (DFP).
- Lenvenberg-Marquardt
- Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Méthode de quasi-Newton : *algorithme de Davidson-Fletcher-Powell (DFP)*

Idée: On estime non pas le Hessien, mais directement son inverse

Paramètres:

- S^0 : matrice définie positive

$k = 0$

Tant que *Critère d'arrêt* non satisfait

$$d^k = -S^k \nabla f\left(x^k\right)$$

Choisir α^k par une méthode de calcul de pas (Armijo, etc.)

$$x^{k+1} = x^k + \alpha^k d^k$$

$$\gamma^k = \nabla f\left(x^{k+1}\right) - \nabla f\left(x^k\right)$$

$$\delta^k = \alpha^k d^k$$

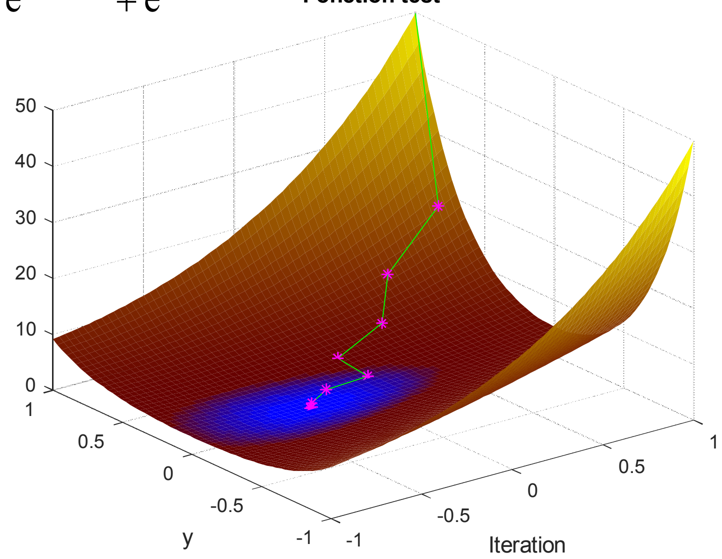
$$S^{k+1} = S^k + \frac{\delta^k \left(\delta^k\right)^T}{\left(\delta^k\right)^T \gamma^k} - \frac{S^k \gamma^k \left(S^k \gamma^k\right)^T}{\left(\gamma^k\right)^T S^k \gamma^k}$$

Méthode de quasi-Newton : *algorithme de Davidson-Fletcher-Powell (DFP)*

```
IterAlpha=0;
Sk=eye(size(x,1));
while ended==0
    dfk=df(x(:,k));
    dk=-Sk*dfk;
    alpha=Armijo(f2,x(:,k),dk,dfk,alpha0,beta,gamma);

    x(:,k+1)=x(:,k)+alpha*dk;
    Gammak=df(x(:,k+1))-dfk;
    deltak=alpha*dk;
    Sk=Sk+(deltak*deltak')/(deltak'*Gammak)-((Sk*Gammak)*(Sk*Gammak)')/(Gammak'*Sk*Gammak);

    ended=(k>IterMax)||...
        norm(df(x(:,k)))<EpsGrad||...
        norm(x(:,k)-x(:,k+1))<Tolx;
    k=k+1;
end
```

$$f(x,y)=e^{x-3y-\frac{1}{10}}+e^{x+3y-\frac{1}{10}}+e^{-x-\frac{1}{10}}$$

DFP+ARMIJO
14 itérations boucle principale
52 appels de f



Méthode de quasi-Newton : *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*

Paramètres:

- S^0 : matrice définie positive

$k = 0$

Tant que *Critère d'arrêt* non satisfait

$$d^k = -S^k \nabla f\left(x^k\right)$$

Choisir α^k par une méthode de calcul de pas (Armijo, etc.)

$$x^{k+1} = x^k + \alpha^k d^k$$

$$\gamma^k = \nabla f\left(x^{k+1}\right) - \nabla f\left(x^k\right)$$

$$\delta^k = \alpha^k d^k$$

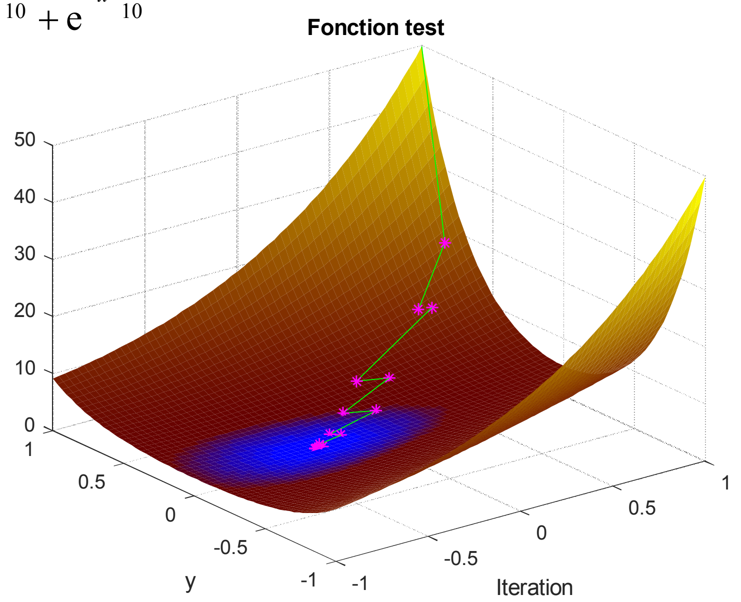
$$S^{k+1} = \left(I - \frac{\delta^k \left(\gamma^k\right)^T}{\left(\delta^k\right)^T \gamma^k}\right) S^k \left(I - \frac{\delta^k \left(\gamma^k\right)^T}{\left(\delta^k\right)^T \gamma^k}\right) + \frac{\delta^k \left(\delta^k\right)^T}{\left(\delta^k\right)^T \gamma^k}$$

Méthode de quasi-Newton : *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*

```
IterAlpha=0;
n=size(x,1);
Sk=eye(n);
while ended==0
    dfk=df(x(:,k));
    dk=-Sk*dfk;
    alpha=Armijo(f2,x(:,k),dk,dfk,alpha0,beta,gamma);

    x(:,k+1)=x(:,k)+alpha*dk;
    Gammak=df(x(:,k+1))-dfk;
    deltak=alpha*dk;
    Sk=(eye(n)-deltak*Gammak'/(deltak'*Gammak))*Sk*(eye(n)-deltak*Gammak'/(deltak'*Gammak))+deltak*deltak'/(deltak'*Gammak);
    fxk=fxkp1;
    fxkp1=f(x(:,k+1));

    ended=(k>IterMax)||...
        norm(df(x(:,k)))<EpsGrad||...
        norm(x(:,k)-x(:,k+1))<Tolx;
    k=k+1;
end
```

$$f(x,y)=e^{x-3y-\frac{1}{10}}+e^{x+3y-\frac{1}{10}}+e^{-x-\frac{1}{10}}$$

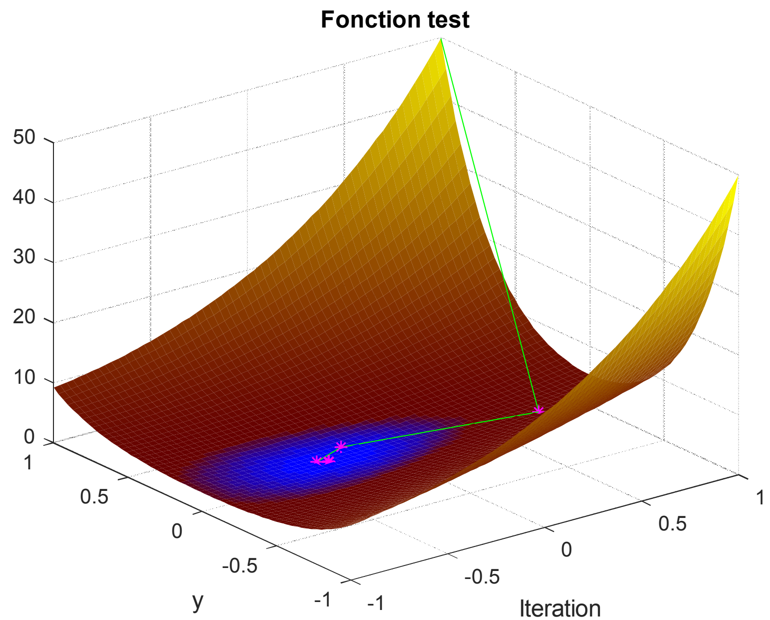
BFGS+ARMIJO
22 itérations boucle principale
105 appels de f



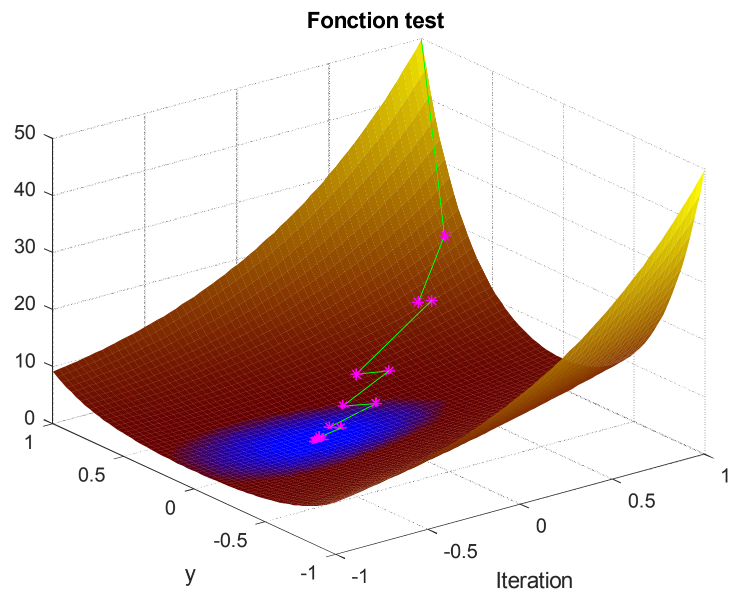
Méthode de quasi-Newton : *Broyden-Fletcher-Goldfarb-Shanno (BFGS)*

En pratique, l'algorithme BFGS pour la direction + Règles de Wolfe pour le choix du pas est souvent très efficace

BFGS+Wolfe
10 itérations boucle principale
56 appels de f



BFGS+ARMIJO
22 itérations boucle principale
105 appels de f



Résumé

Algorithme d'optimisations: $x^{k+1} = x^k + \alpha^k d^k$

- Trouver un pas α^k
- Trouver une direction d^k

Algorithme de choix du pas	Performances	Facilité de mise en œuvre
Pas optimal	++	Impossible sauf sur quelques cas simples
Pas constant	----	++++
Backtracking	+	+++
Armijo	++	++
Armijo-Golstein	++	+
Wolfe	++++	--

Algorithme de choix de la direction	Performances	Facilité de mise en œuvre
Plus forte pente (Gradient)	+	+++ Nécessite le gradient
Newton	++	Nécessite un Hessien définit positif
DFP	+++	+
BFGS	++	+++

1) Contexte
2) Algorithme à base de Gradient
3) Méthode de Newton & quasi-Newton
4) Méthode « Régions de confiance » (Trust region)

Référence utilisée pour ce cours: Tamas Terlaky, Advanced Optimization Lab., CAS, McMaster 9

Régions de confiance

Autour de x^k, la fonction f peut être approché (localement) par une quadratique

$$q(x) \simeq f\left(x^k\right) + \nabla f\left(x^k\right)^T \left(x - x^k\right) + \frac{1}{2}\left(x - x^k\right)^T \nabla^2 f\left(x^k\right)\left(x - x^k\right)$$

On suppose que l'approximation est valide pour une région dite de confiance:

$$\left\| x - x^k \right\| < \delta^k$$

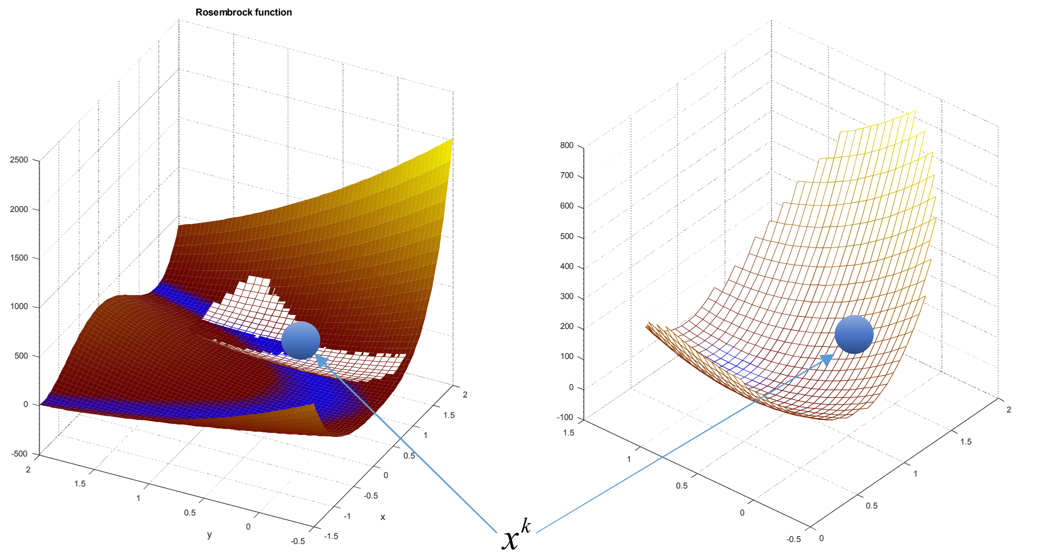
La forme dépend de la norme choisie :

- $\|\ \|_2$: cercle
- $\|\ \|_\infty$: carré

Dans le cours, on utilise $\|\ \|_2$

Exemple : fonction de Rosembrock

$$f(x,y) = (1-x)^2 + 100\left(y - x^2\right)^2$$



Fonction

Modèle local quadratique

Régions de confiance

<u>Algorithme de base :</u>

Paramètres $\eta_1 < \eta_2 < 1$

$\gamma_{red} < 1 \quad \gamma_{aug} > 1$

Init : $k = 0 \quad x^0 = x_0 \quad \delta^0 = 1$

Tant que Critère non satisfait

$$s^k = \underset{\|s^k\| \leq \delta^k}{\arg\min}\, q\left(x^k + s^k\right) = f\left(x^k\right) + \nabla f\left(x^k\right)^T s^k + \frac{1}{2}\left(s^k\right)^T \nabla^2 f\left(x^k\right) s^k$$

$$Ratio = \frac{f\left(x^k + s^k\right) - f\left(x^k\right)}{q\left(x^k + s^k\right) - q\left(x^k\right)}$$

Variation de la fonction

Variation du modèle

Mesure la confiance que l'on peut avoir dans le modèle

Si $Ratio < \eta_1$ alors

$\delta^k = \delta^k \cdot \gamma_{red}$ *Echec, on réduit la taille de la région*

Sinon

$x^{k+1} = x^k + s^k$ *Réussite, on accepte la solution obtenue*

Si $Ratio > \eta_2$ alors

$\delta^{k+1} = \delta^k \cdot \gamma_{aug}$ *« Grande » Réussite, on augmente la taille de la région de confiance*

Sinon

$\delta^{k+1} = \delta^k$ *Réussite « modeste »: On garde la même taille de région*

Fin si

$k = k + 1$

Fin si

Fin tant que

Régions de confiance

```
clear all;
close all;
clc;

NoFonction=3;
ChoixFonction;

x0=[0; 0.5];

% Affichage de la fonction
X=linspace(xmin,xmax,61);
Y=linspace(ymin,ymax,62);
Z=zeros(length(Y),length(X));
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f([X(i);Y(j)]);
    end
end

figure;
surf(X,Y,Z,'EdgeColor','none');
hold on;
xlabel('x');
ylabel('y');
title(Fname)
%colorbar;
colormap(map)
hold on;
plot3(x0(1),x0(2),f(x0),'bo');

% Criteres d'arret
Tolf=1e-6;
Tolx=1e-9;
IterMax=100;
```

```
while ended==0
    xk=x(:,k); % sol courante
    fxk=f(xk); % valeur de la fonction
    g=df(xk);  % gradient
    H=d2f(xk); % hessien

    % Modèle au point xk
    q=@(x2) fxk+g'*(x2-x(:,k))+0.5*(x2-x(:,k))'*H*(x2-x(:,k));

    % Résolution du pb Trust Region
    s(:,k)=xxxxxxxxxxxxxxxxxxxxx

    xkp1=xk+s(:,k); % prochain point ?

    plot3(xkp1(1),xkp1(2),f2(xkp1),'r*');
    plot3([xkp1(1) xk(1)],[xkp1(2) xk(2)],[f2(xkp1) fxk],'g');

    % Amélioration sur le Modele et la fonction
    ModeleDiff=q(xk+s(:,k))-q(xk);
    FonctionDiff=f(xkp1)-fxk;
    Ratio=FonctionDiff/ModeleDiff;

    % Gestion de la taille de la région
    if Ratio<nu1
        % Echec
        Delta(k)=Delta(k)*Gamma1;
    else
        % Réussite
        x(:,k+1)=xkp1;
        if Ratio>nu2
           % Grande réussite
            Delta(k+1)=Delta(k)*Gamma2;
        else
            % Réussite modeste
            Delta(k+1)=Delta(k);
        end
```

Appeler ici la fonction qui calcule s

```
        fxkp1=f(x(:,k+1));
            ended=(k>IterMax)||...
                norm(fxkp1-fxk)<Tolf||...
                norm(x(:,k)-x(:,k+1))<Tolx;
             k=k+1;

    end
end
```

Régions de confiance

<u>Résolution de l'optimisation du problème contraint:</u>

Intérêt de la contrainte: il existe toujours une solution

Nécessite de savoir gérer les problèmes avec contraintes (cf prochain cours)

Le problème d'optimisation :

$$\min J = f\left(x^k\right) + \nabla f\left(x^k\right)^T s + \frac{1}{2} s^T \nabla^2 f\left(x^k\right) s$$

$$\text{sous la contrainte : } \|s\| \leq \delta$$

A pour solution: $s(\mu) = -\left(\nabla^2 f\left(x^k\right) + \mu I\right)^{-1} \left(\nabla f\left(x^k\right)\right)^T$

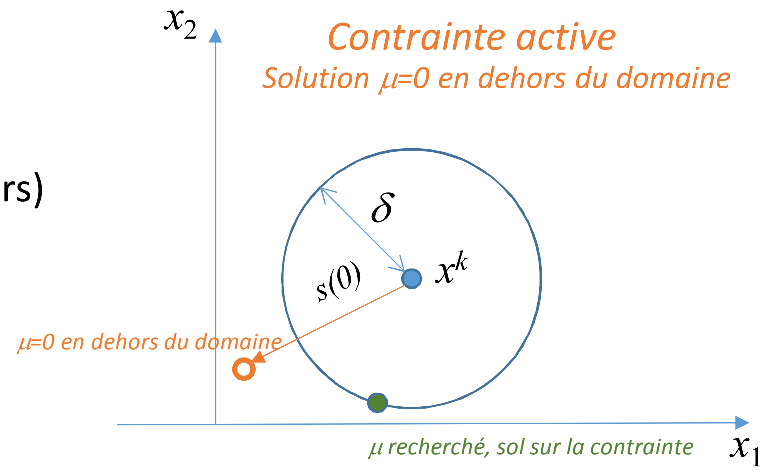
Si $\|s(0)\| > \delta$ alors

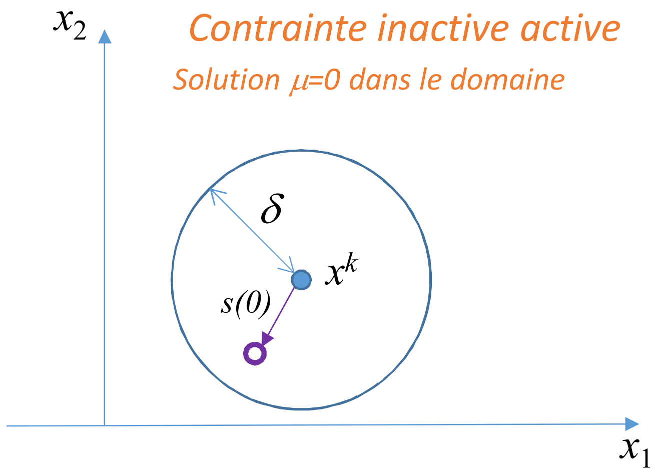
μ est tel que $\|s(\mu)\| = \delta$ *Contrainte active*

Sinon

μ=0 *Contrainte non active*

Fsi





Régions de confiance

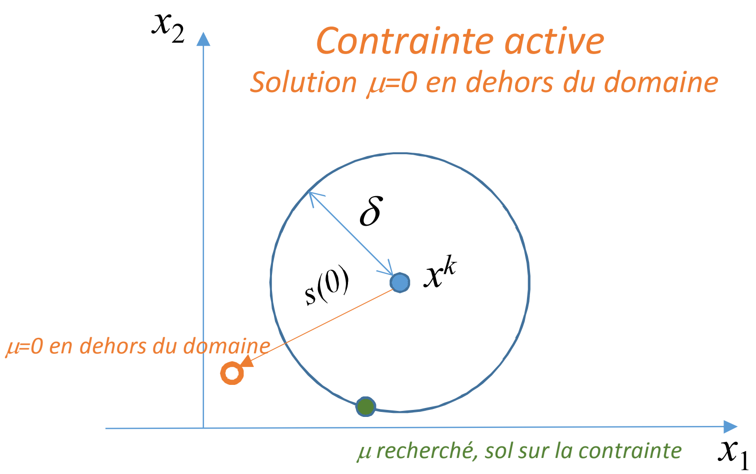
Pourquoi les algorithmes à base de région de confiance sont performants ?

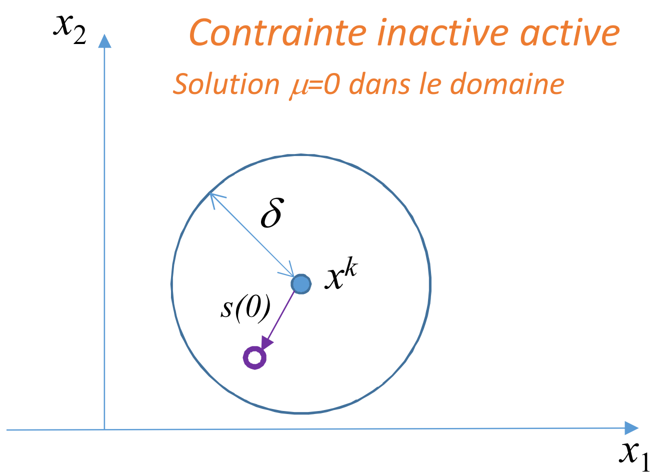
<u>La direction de recherche change avec la contrainte</u>: quand la taille de la région change, la direction de recherche change (elle évolue entre la direction donnée par la méthode de descente maximale et celle de Newton)
=> Permet d'éviter un point de scelle local par exemple

$$s(\mu) = -\left(\nabla^2 f\left(x^k\right) + \mu I\right)^{-1} \left(\nabla f\left(x^k\right)\right)^T$$

$\mu = 0$: $s(0) = -\nabla^2 f\left(x^k\right)^{-1} \left(\nabla f\left(x^k\right)\right)^T$ => Newton

$\mu \to +\infty$: $s(\mu) \to -\left(\nabla f\left(x^k\right)\right)^T$ => Plus forte pente





Régions de confiance : *Méthode de Cauchy (plus forte pente)*

<u>Méthode de Cauchy:</u> 1) On avance en suivant la plus forte pente
2) On s'arrête sur la contrainte

Problème non contraint: $s^{uc} = \arg\min_{s} q\left(x^k + s\right)$

$$q\left(x^k + s\right) = f\left(x^k\right) + {}^T s + \frac{1}{2} s^T H s \qquad H = \nabla^2 f\left(x^k\right) \quad g = \nabla f\left(x^k\right)$$

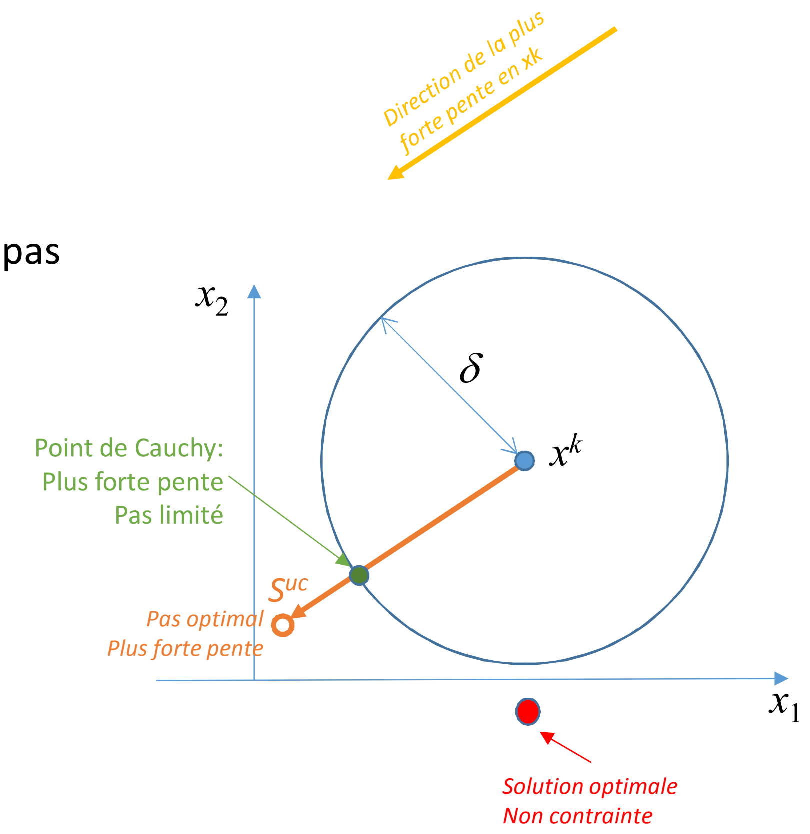
Direction de la plus forte pente: $s = -t \cdot g$ Avec t la longueur du pas

$$\Rightarrow q\left(x^k + s\right) = f\left(x^k\right) - t g^T g + \frac{t^2}{2} g^T H g$$

Choix du pas t optimal:

$$\frac{\partial q\left(x^k + s\right)}{\partial t} = -g^T g + t g^T H g = 0 \qquad t = \frac{g^T g}{g^T H g}$$

$$\Leftrightarrow s^{uc} = -\frac{g^T g}{g^T H g} g$$



Régions de confiance : *Méthode de Cauchy (plus forte pente)*

<u>Méthode de Cauchy:</u>

1) On avance en suivant la plus forte pente
2) On s'arrête sur la contrainte

Algorithme

Si $g^T H g > 0$ alors

La fonction est localement convexe

$$t = \frac{g^T g}{g^T H g} \qquad s^{uc} = -t \cdot g$$

$$longeur = \left\| s^{uc} \right\| = \sqrt{\left(s^{uc}\right)^T \left(s^{uc}\right)} = t \cdot \left\| g \right\|$$

Si longueur<δ alors

$s = s^{uc}$ *Sol non contrainte est ok*

Sinon

$$t = \delta$$

$$s = -t \cdot \frac{g}{\left\| g \right\|}$$

On doit limiter la longueur du pas

Fin si

Sinon

La fonction n'est pas localement convexe

$$t = \delta$$

$$s = -t \cdot \frac{g}{\left\| g \right\|}$$

On prends le plus grand pas possible

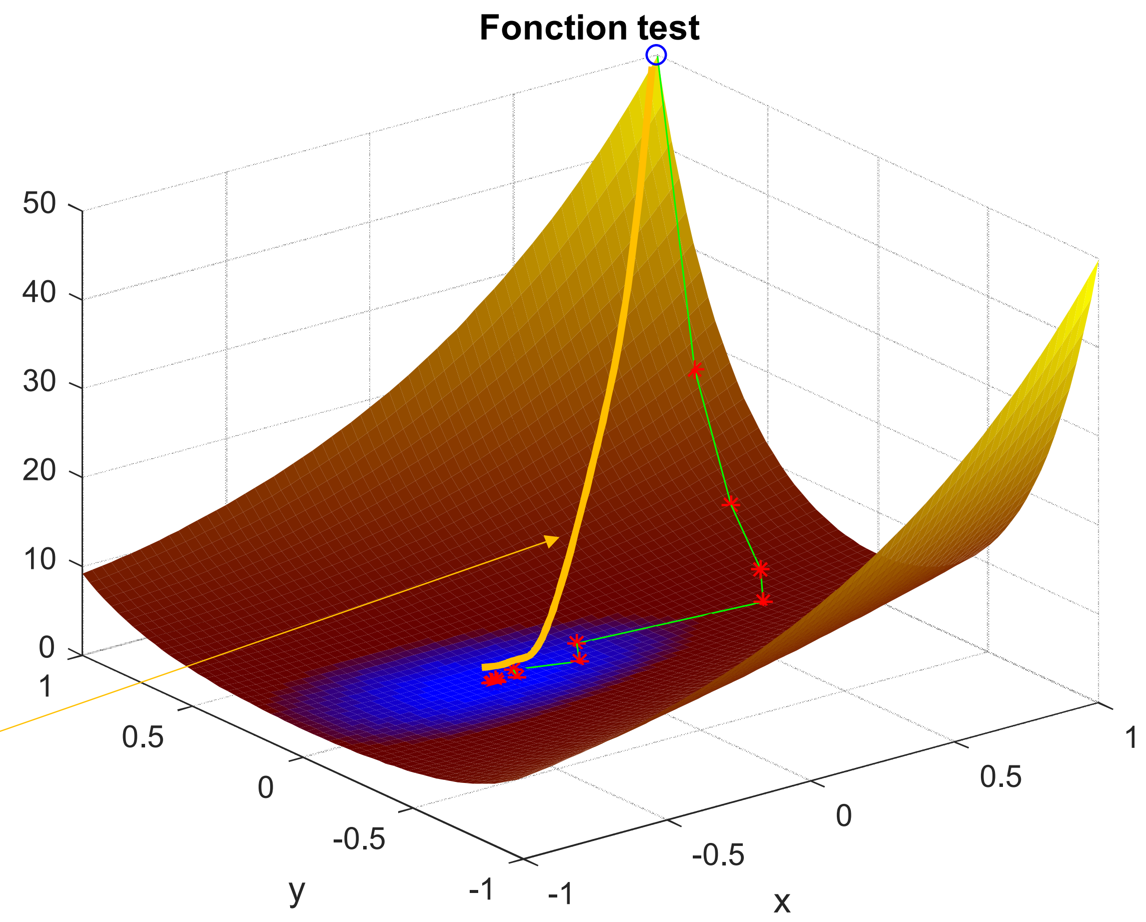
Fin si

```
function s=TR_Cauchy(g,H,Delta)
  gHg = g'*H*g;

  if gHg>5*eps,
    % La fonction est localement convexe
    % Choix de la plus forte descente
    t=(g'*g)/gHg;
    s_uc = -t*g;
    longueur = norm(s_uc); % Norme 2
    if longueur <= Delta,
       % Solution retenue
        s=s_uc;
    else
        % On sature la longueur du pas
       s= -Delta*g/norm(g);
    end
  else
    % Modèle localement non convexe:
    % SD minimizer lies on TR boundary, in direction of -g'
    s= -Delta*g/norm(g); % On avance dans la direction de g, longueur : delta
  end

end
```

Régions de confiance : *Méthode de Cauchy (plus forte pente)*

<u>Exemple:</u> $f(x,y) = e^{x-3y-\frac{1}{10}} + e^{x+3y-\frac{1}{10}} + e^{-x-\frac{1}{10}}$

Trust Region + Cauchy
17 itérations boucle principale
49 appels de f



Intérêt :
Méthode très robuste en général

Inconvénient :
On reste sur la direction de la plus forte pente
$\Rightarrow$ Peu mieux faire

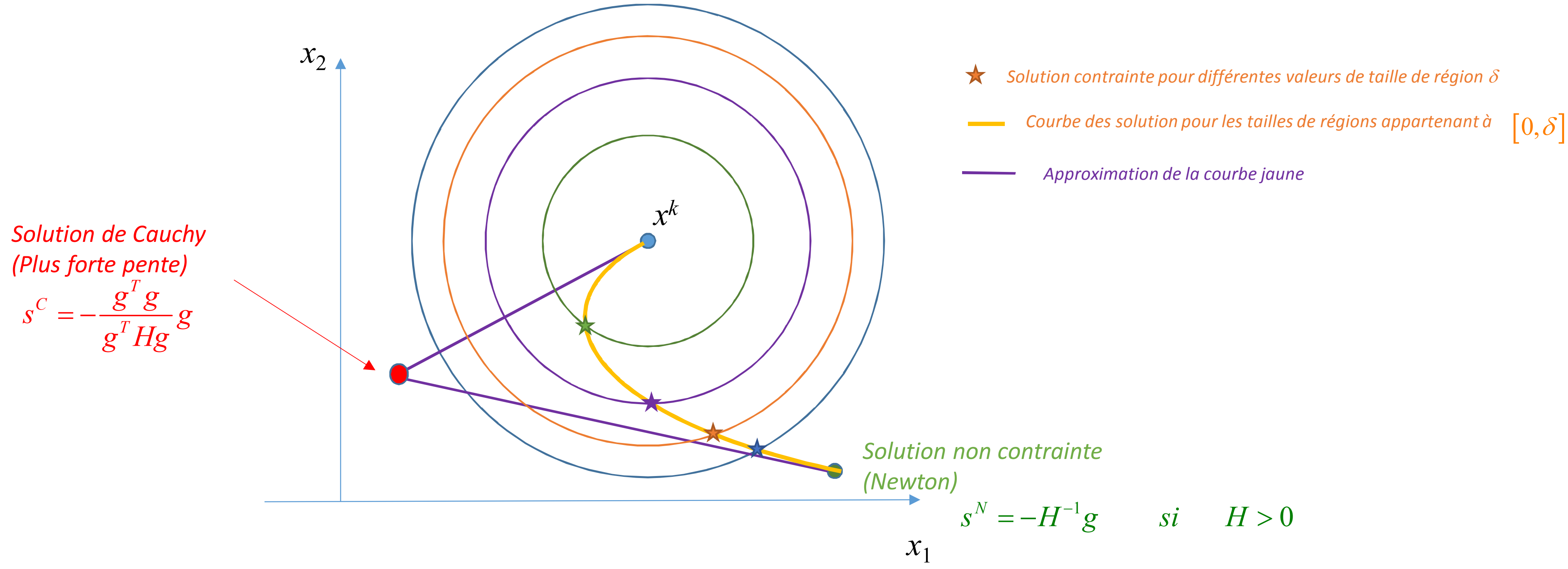
Une descente vers le minimum serait potentiellement plus efficace

Régions de confiance : *Méthode « Dogleg »*

$$\min J = f\left(x^k\right) + g^T s + \frac{1}{2} s^T H s \qquad H = \nabla^2 f\left(x^k\right) \quad g = \nabla f\left(x^k\right)$$

sous la contrainte : $\|s\| \le \delta$

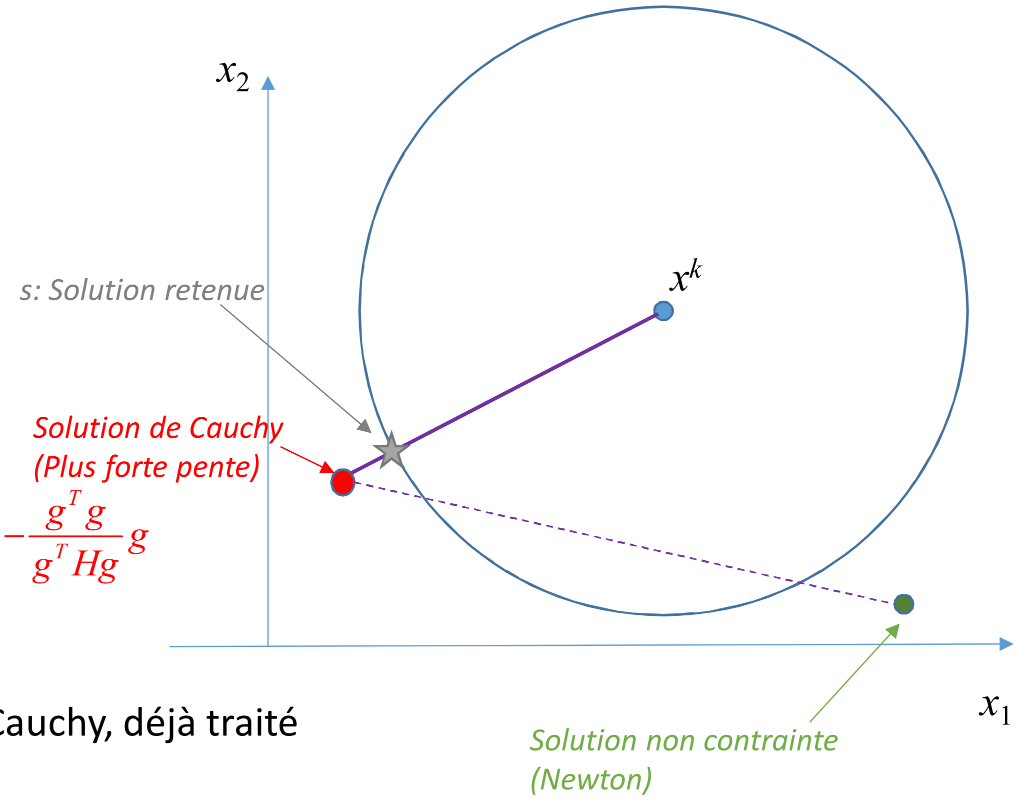
PB : Calculer l'intersection de la courbe jaune avec la limite de la région est impossible



Régions de confiance : *Méthode « Dogleg »*

$$\min J = f\left(x^k\right) + g^T s + \frac{1}{2} s^T H s \qquad H = \nabla^2 f\left(x^k\right) \quad g = \nabla f\left(x^k\right)$$

sous la contrainte : $\|s\| \leq \delta$



$$s^C = -\frac{g^T g}{g^T H g} g$$

$$s^N = -H^{-1} g \qquad si \qquad H > 0$$

<u>On approche la courbe par 2 segments de droites</u>

=> Calcul de l'intersection entre une droite et un cercle facile

<u>Cas n° 1</u> : Intersection entre $[x^k, s^c]$ et le cercle : C'est le point de Cauchy, déjà traité

$$s = -\delta \cdot \frac{g}{\|g\|}$$

Direction du gradient (normalisée)

Pas maximal autorisé: δ

Régions de confiance : *Méthode « Dogleg »*

$$\min J = f\left(x^k\right) + g^T s + \frac{1}{2} s^T H s \qquad H = \nabla^2 f\left(x^k\right) \quad g = \nabla f\left(x^k\right)$$

sous la contrainte : $\|s\| \le \delta$

<u>Cas n° 2</u> : Intersection entre le segment $[s^N, s^c]$ et le cercle

Equation d'un point sur le segment $[s^N, s^c]$:

$$p(t) = x^k + s^c + t\left(s^N - s^c\right) \qquad 0 < t \le 1$$

On cherche l'intersection avec le cercle, donc: $s^c + t(s^N - s^c)$ $\quad s^C = -\dfrac{g^T g}{g^T H g} g$

$$\left\| p(t) - x^k \right\|^2 = \delta^2$$

$$\left(s^c + t\left(s^N - s^c\right)\right)^T \left(s^c + t\left(s^N - s^c\right)\right) = \delta^2$$

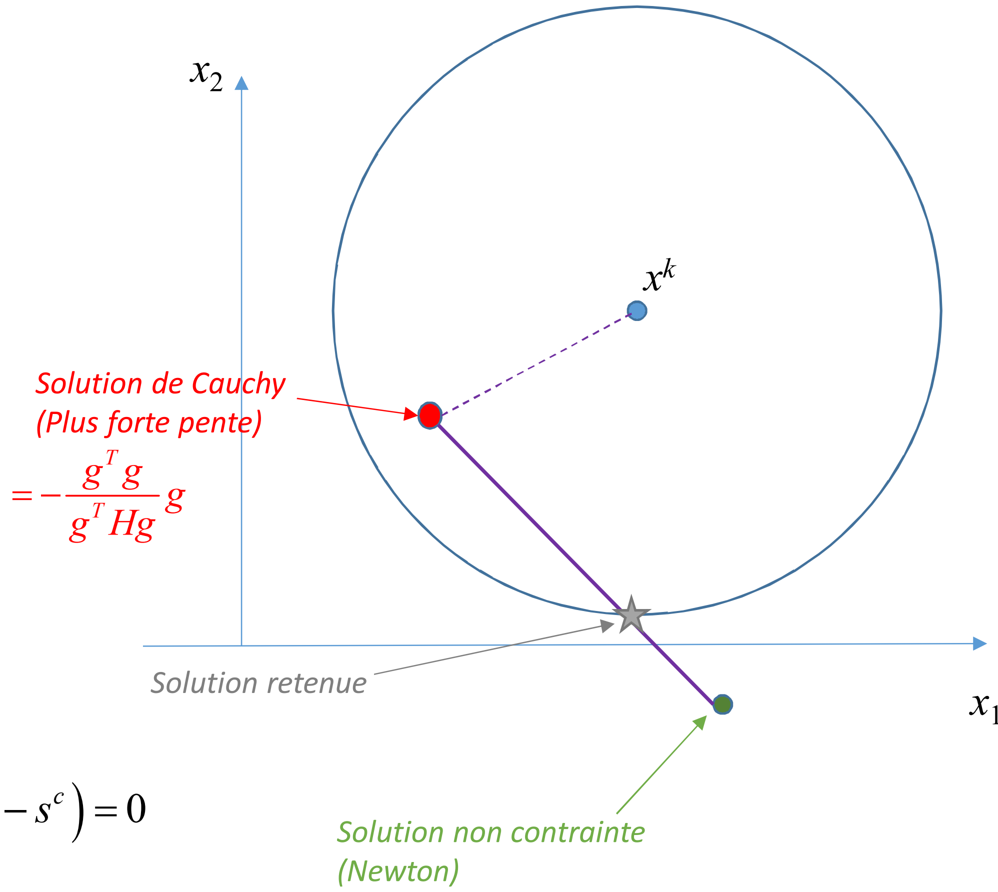
$$\left(\left(s^c\right)^T + t\left(s^N - s^c\right)^T\right)\left(s^c + t\left(s^N - s^c\right)\right) = \delta^2$$

$$\left(s^c\right)^T s^c - \delta^2 + t\left[\left(s^N - s^c\right)^T s^c + \left(s^c\right)^T \left(s^N - s^c\right)\right] + t^2 \left(s^N - s^c\right)^T \left(s^N - s^c\right) = 0$$

$$a + bt + ct^2 = 0$$

2 racines, on choisit celle qui appartient au segment

$$s = s^c + t\left(s^N - s^c\right)$$



$s^N = -H^{-1} g \qquad si \qquad H > 0$

Régions de confiance : *Méthode « Dogleg »*

Algorithme:

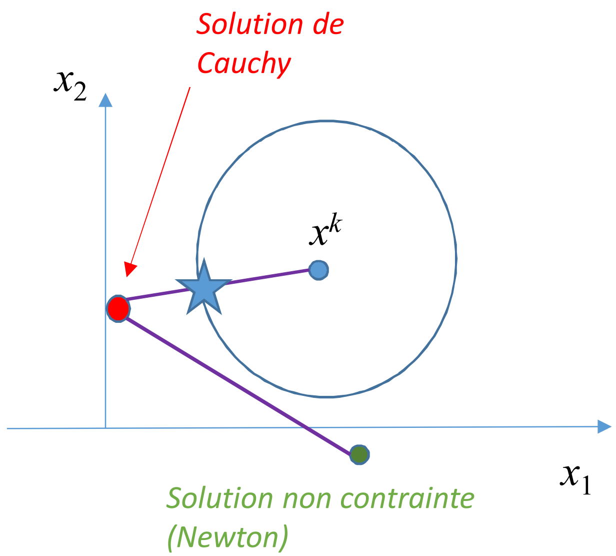
1) Vérifier que la méthode de Newton (non contrainte) améliore la solution

$\Rightarrow$ Permet de détecter les cas ou le Hessien n'est pas définit positif

$\Rightarrow$ Dans ce cas, utiliser la méthode de Cauchy

Sol de Newton: $s^N = -H^{-1} \cdot g$

Variation du modèle: $q\left(x^{k+1}\right) - q\left(x^k\right) = g^T s^N + \frac{1}{2} g^T H g$



Régions de confiance : *Méthode « Dogleg »*

Algorithme:

2) Si $q\left(x^{k+1}\right)-q\left(x^{k}\right)<0$ alors

Si newton est dans le domaine ($\left\|s^{N}\right\|\leq\delta$) => on garde cette solution

Sinon

On calcule la solution de Cauchy (non contrainte) : $s^{c}=-\frac{g^{T}g}{g^{T}Hg}\cdot g$

Si Cauchy en dehors de la contrainte alors

on s'arrête en bordure de région

Sinon

=>Cauchy en dehors de la contrainte:

On choisit à l'intersection du cercle et du segment $[s^c,s^N]$

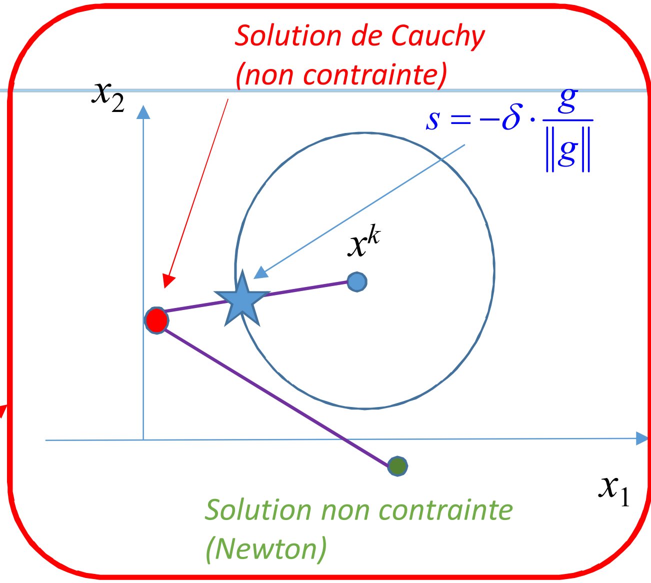
$$t=\frac{-b+\sqrt{b^{2}-4ac}}{2a}$$

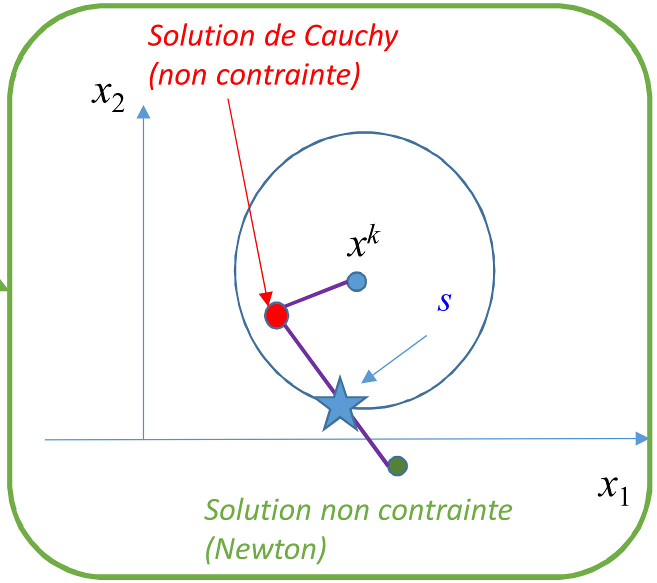
$$s=s^{c}+t\left(s^{N}-s^{c}\right)$$

$$a=\left(s^{N}-s^{c}\right)^{T}\left(s^{N}-s^{c}\right)$$

$$b=\left(s^{N}-s^{c}\right)^{T}s^{c}+\left(s^{c}\right)^{T}\left(s^{N}-s^{c}\right)$$

$$c=\left(s^{c}\right)^{T}s^{c}-\delta^{2}$$





Régions de confiance : *Méthode « Dogleg »*

```
function s=TR_Dodleg(g,H,Delta)
% Calcul de la solution uncontrainte avec Newton
sN=-H\g;

% Evaluation de q(xk+1)-q(x)
DiffQ=g'*sN+0.5*sN'*H*sN;

if DiffQ<0
    % La sol de Newton améliore la solution
    if norm(sN)<Delta
        % La solution est acceptée
        s=sN;
    else
        % La sol de Newton est en dehors de la région
        % On calcule la solution de Cauchy
        sC = -(g'*g)/( g'*H*g)*g;
        if norm(sC)>Delta
            % La sol de cauchy est en dehors
            % On limite la taille du pas dans la direction de descente
            s= -Delta*g/norm(g);
        else
            % La sol de Newton est dehors, celle de Cauchy dedans
            % On cherche l'intersection du cercle et du segment
            DiffS=sN-sC;
            a=DiffS'*DiffS;
            b=DiffS'*sC+sC'*DiffS;
            c=sC'*sC-Delta^2;
            t=(-b+sqrt(b^2-4*a*c))/2/a;
            s=sC+t*DiffS;
        end
    end
else
  % La solution de Newton n'améliore pas la solution: point de scelle ou
  % H>0
  % On utilise la méthode de Cauchy
  s=TR_Cauchy(g,H,Delta);
end
```

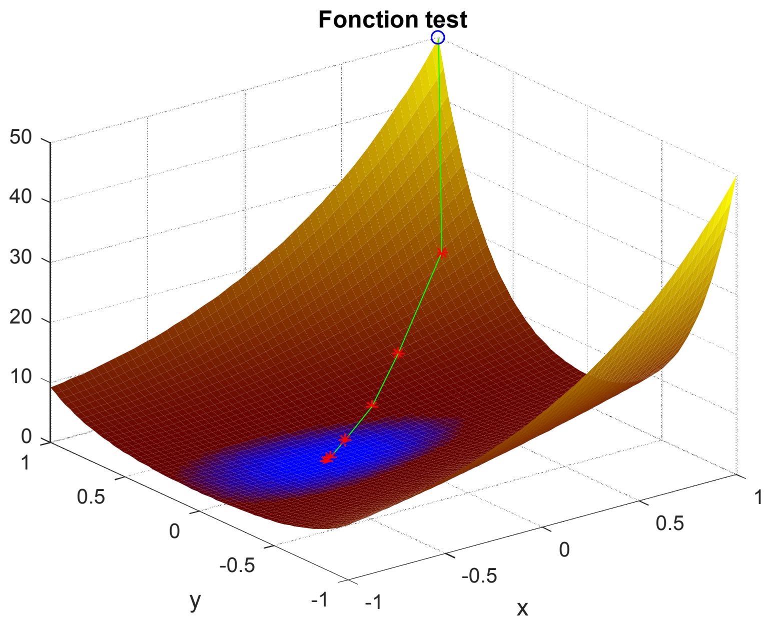
NB: l'algorithme principal reste inchangé

Régions de confiance : *Méthode « Dogleg »*

<u>Algorithme</u>

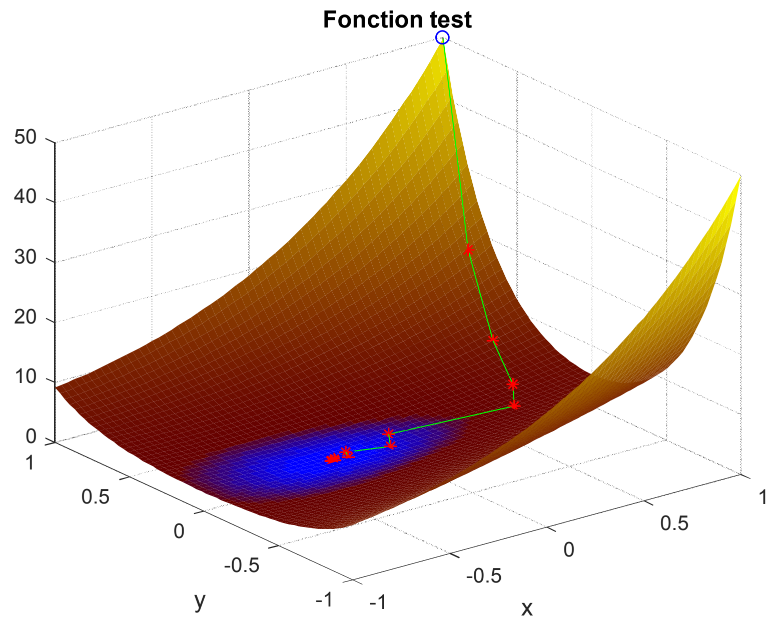
$$f(x,y)=e^{x-3y-\frac{1}{10}}+e^{x+3y-\frac{1}{10}}+e^{-x-\frac{1}{10}}$$

Trust Region + Dodleg
9 itérations boucle principale
25 appels de f



Dodleg: descent directement vers le minimum
=> Avantage lié à la solution de Newton

Trust Region + Cauchy
17 itérations boucle principale
49 appels de f



Plus forte pente: descend vers la droite alors que le minimum est vers la gauche